CppCon 2022

GPU Accelerated Computing and Optimizations on Cross-Vendor Graphics Cards with Vulkan & Kompute

Alejandro Saucedo

@AxSaucedo

Hello, my name is Alejandro

@AxSaucedo

Engineering Director
Seldon Technologies

Chief Scientist
The Institute for Ethical AI & ML

Governing Council Member-at-Large
Association for Computing Machinery

Alejandro Saucedo
@AxSaucedo

High level Objectives

Parallel Processing

GPU Computing

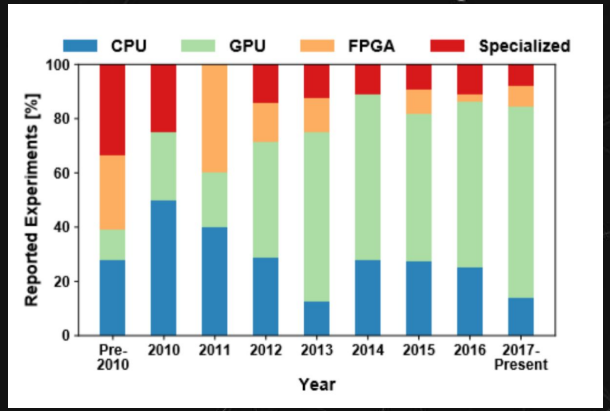
Vulkan SDK

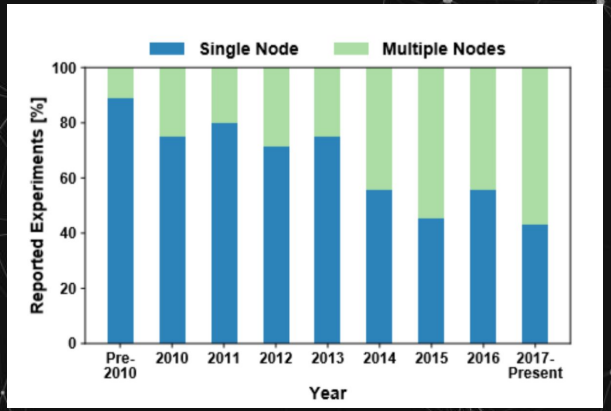
Kompute Framework

Hands on Examples

Why Parallel Processing?

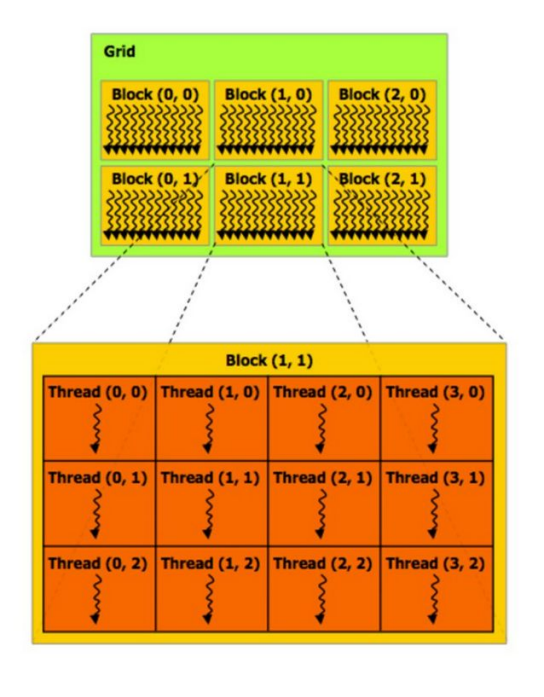
- Functions can often be reduced to highly parallelizable stages (Matrix Mult, ML Layers, etc)
- Micro-batching allows for further parallelization of multiple inputs (eg. cost instead of loss)
- Breaking up fractions of each ensemble comp. across tightly coupled hardware (eg. multi-GPU)

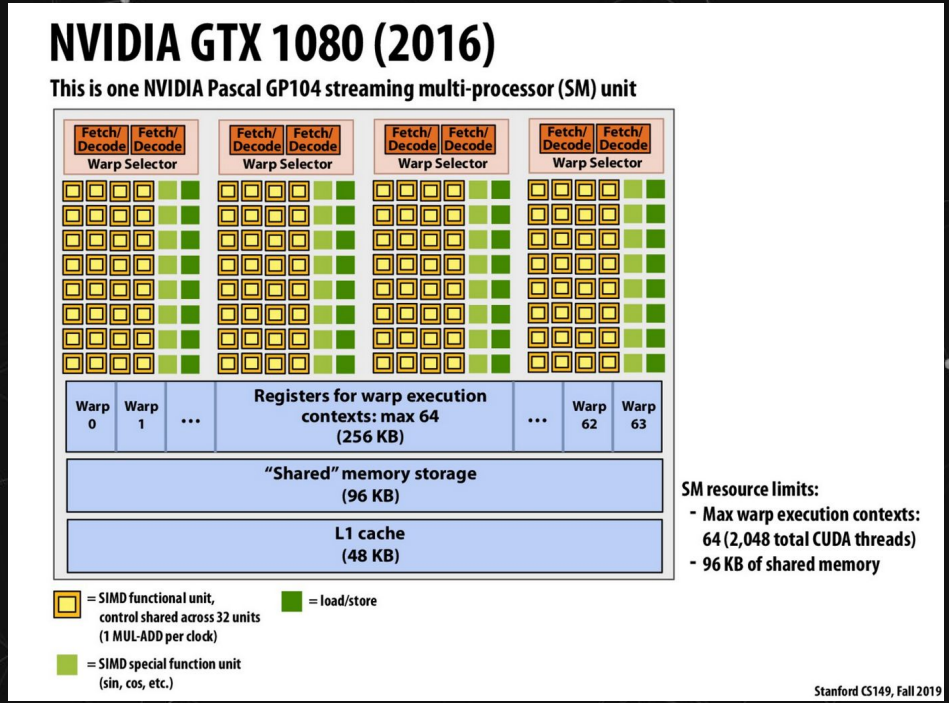




Ben-Nun, Tal, and Torsten Hoefler. "Demystifying parallel and distributed deep learning: An in-depth concurrency analysis." ACM Computing Surveys (CSUR) 52.4 (2019): 1-43.

GPU Compute: Hardware design

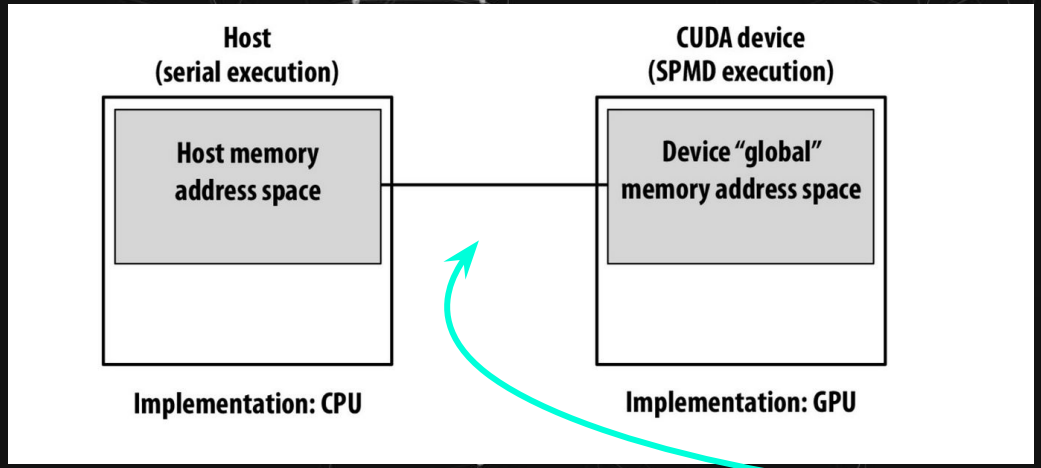




Stanford cs 149 2019 Slides: http://cs149.stanford.edu/fall19/lecture/gpuarch/slide_038

GPU Compute: Memory Model

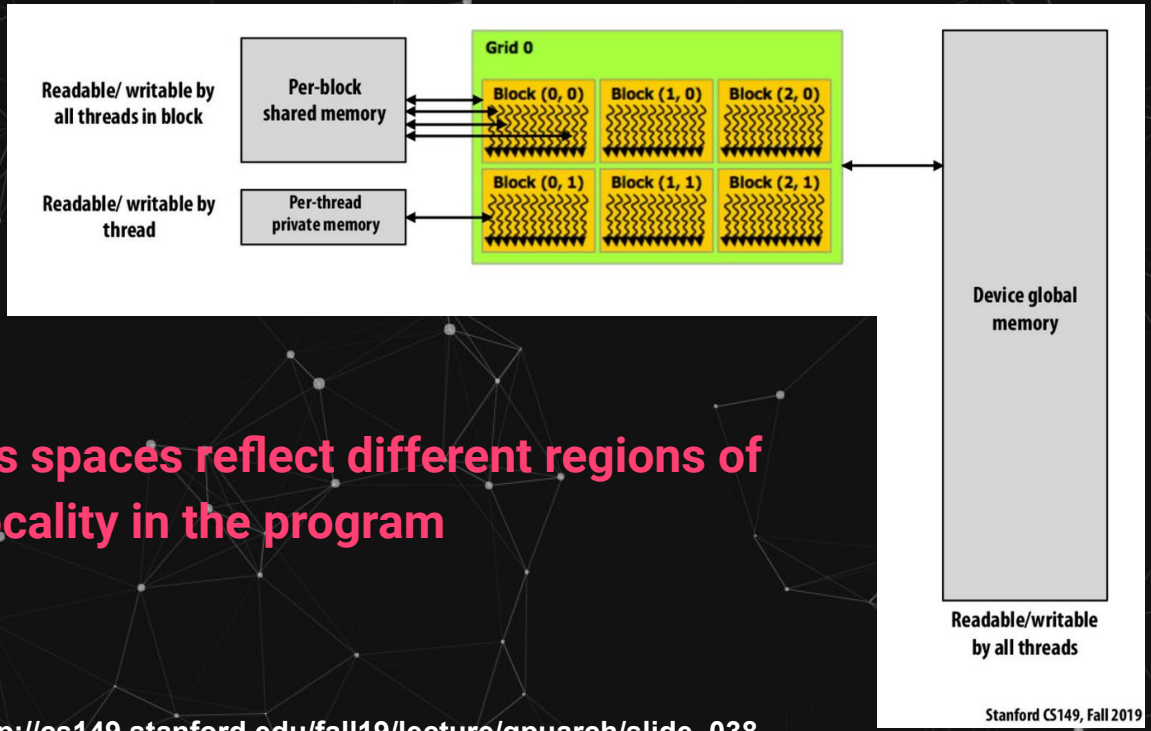
Distinct host and GPU device address spaces



Memory is allocated and copied explicitly

Stanford cs 149 2019 Slides: http://cs149.stanford.edu/fall19/lecture/gpuarch/slide_038

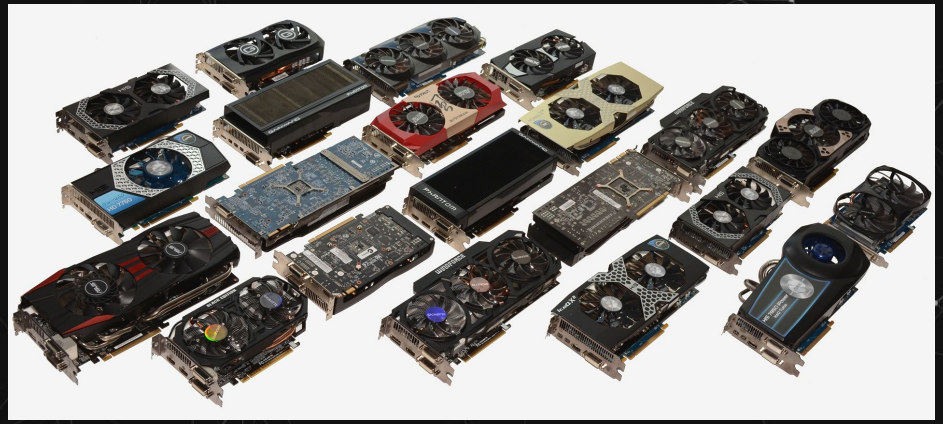
GPU Compute: Memory Model



Different address spaces reflect different regions of locality in the program

Stanford cs 149 2019 Slides: http://cs149.stanford.edu/fall19/lecture/gpuarch/slide_038

Motivations: Heterogeneity



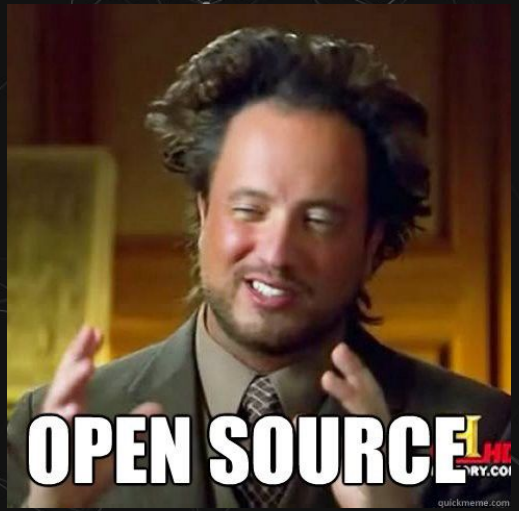
Introducing Vulkan

Created by the Khronos group

The Khronos Group, Inc. is a non-profit member-funded industry consortium, focused on the creation of open standard, royalty free APIs for authoring and accelerated playback of dynamic media on a wide variety of platforms and devices.

Top Vulkan Priorities

1. Performance
2. Interoperability
3. Performance



Oregon State University, SIGGRAPH 2020 Lecture Slides
http://web.engr.oregonstate.edu/~mjb/vulkan/

Khronos Members

Oregon State University, SIGGRAPH 2020 Lecture Slides http://web.engr.oregonstate.edu/~mjb/vulkan/

Vulkan SDK

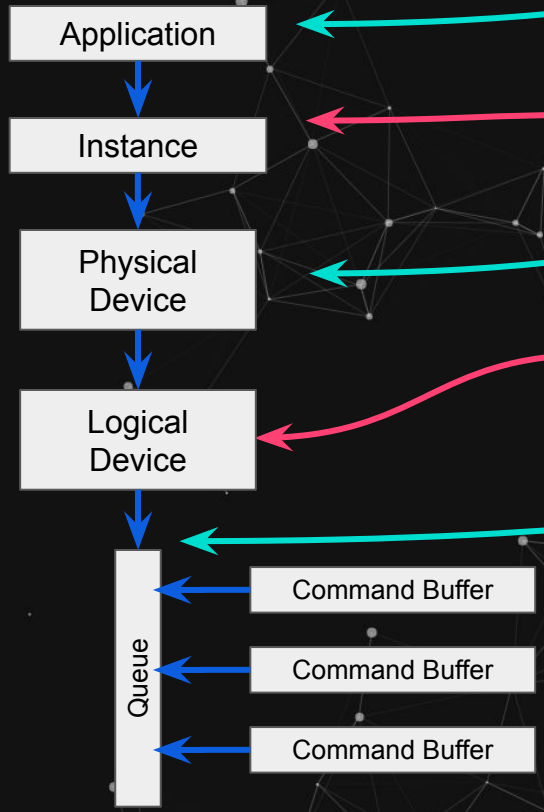
Advantages

- Low level with rich access to components
- C-style API as core interface for developing GPU applications
- A broad range of top players leading the development of the framework
- Highly compatible across different platforms, mobile, and different suppliers

Disadvantages

- Low level with rich access to components
- C-style API as core interface for developing GPU applications
- A broad range of top players leading the development of the framework
- Highly compatible across different platforms, mobile, and different suppliers

Architecture



Vulkan Overarching Application which owns all sub-components

Vulkan can have multiple instances of application

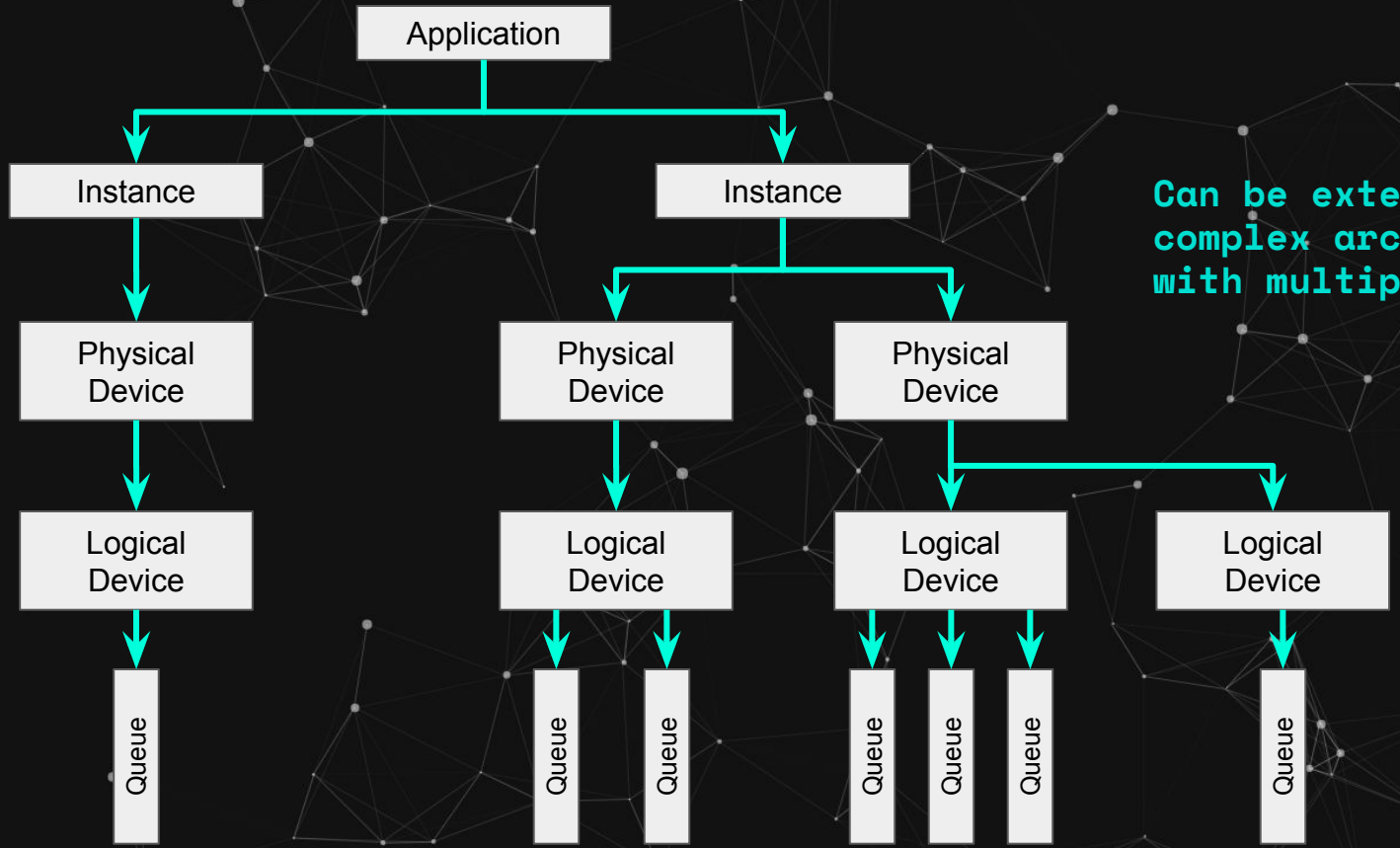
Physical devices specify the underlying GPU card hardware

Logical devices are abstractions for physical devices with set properties

Work gets submitted via queues from CPU and gets picked up and executed from GPU

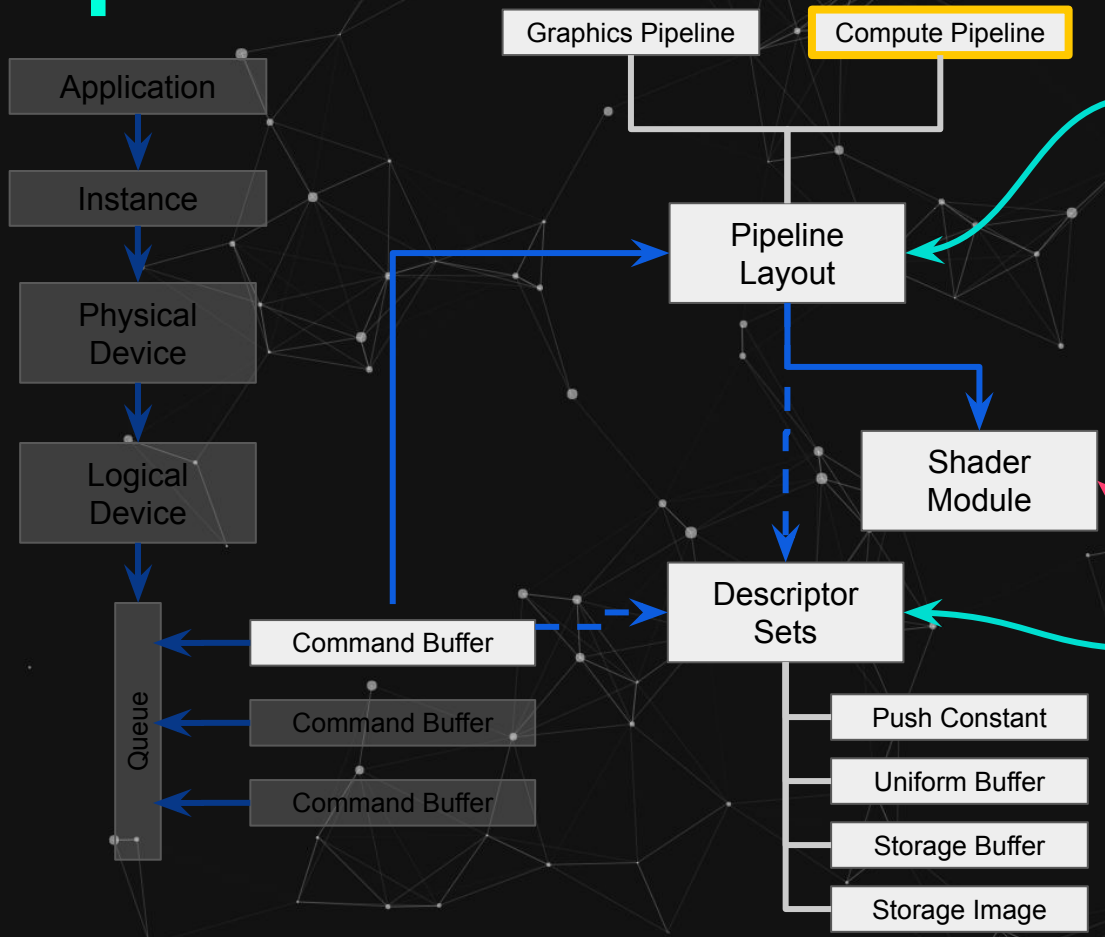
Command buffers specify work to be carried out as an individual unity (move data, run shader, copy data, etc)

Application



Can be extended to complex architectures with multiple devices

Pipeline

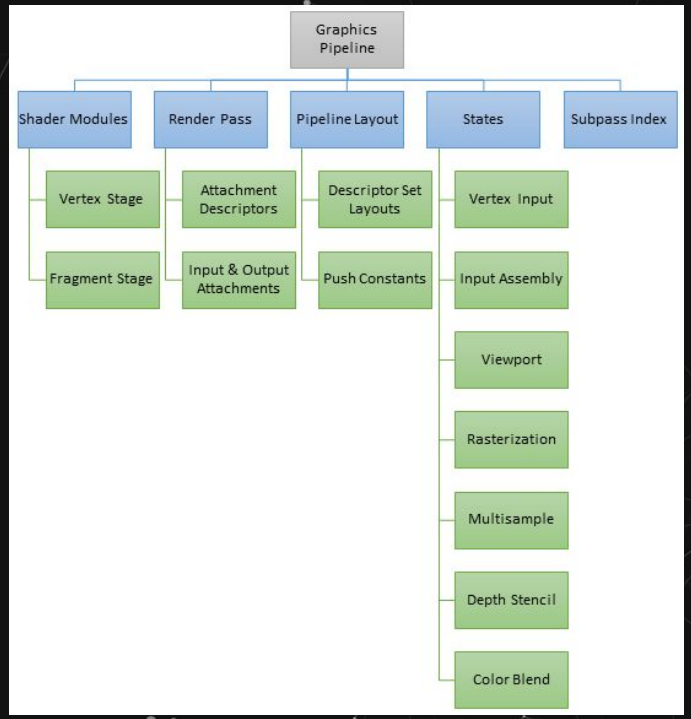


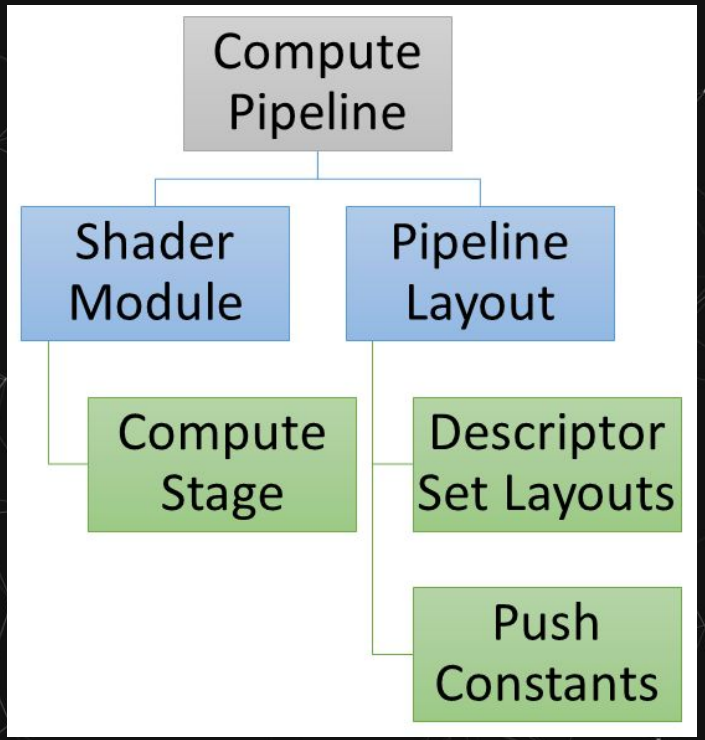
Vulkan Pipeline is core to the "processing" stages, binding the key components on each stage

Shaders consist of the parallel code that runs in the GPU to perform any relevant processing

Descriptors are the components that can be used to bind data from GPU buffers into shaders and pipelines

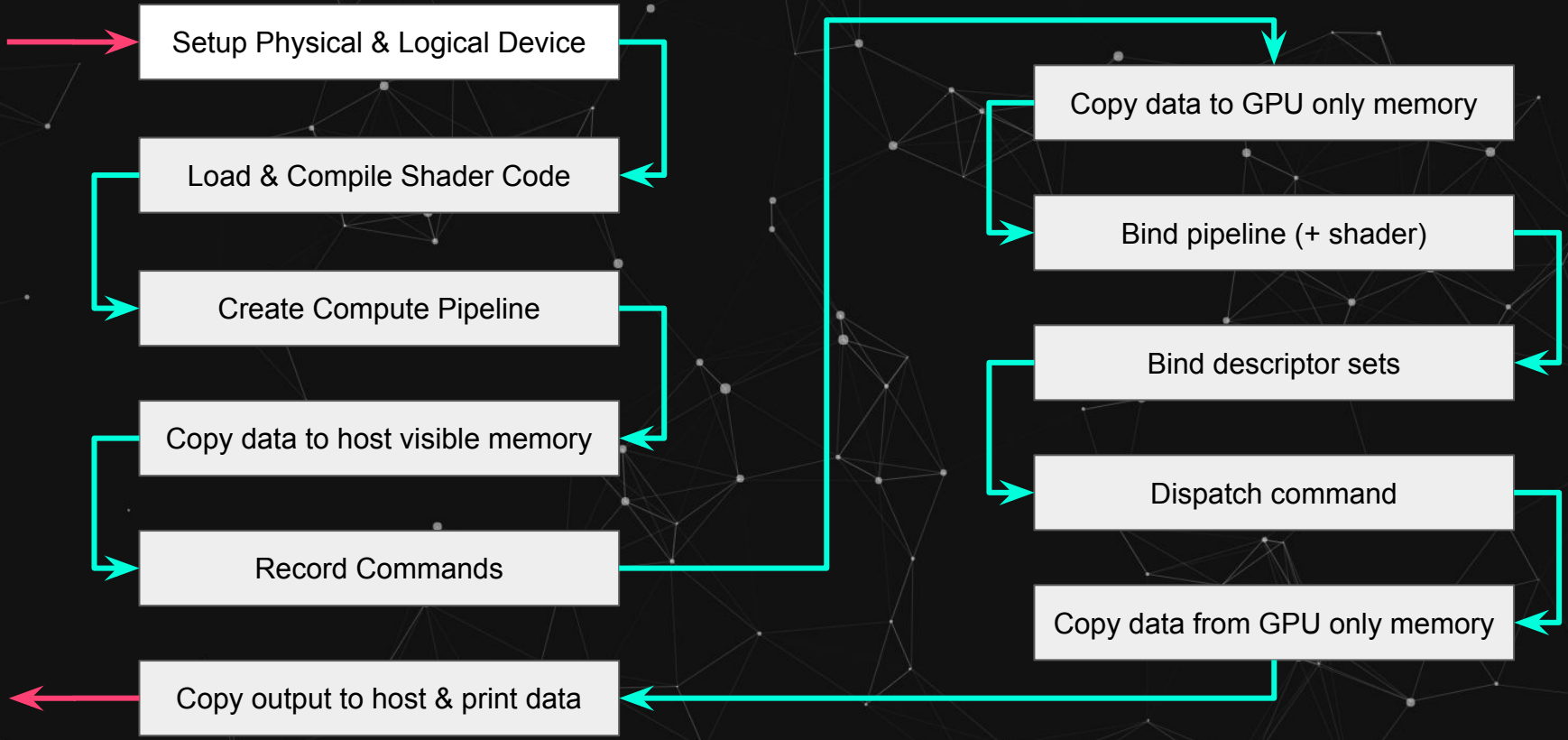
Vulkan Pipeline Types





Vulkan Best Practice for Mobile Developers https://arm-software.github.io/vulkan_best_practice_for_mobile_developers/

The Life of a Vulkan Program



Only takes about

500-2000 lines of code...

Enter Kompute

The General Purpose Vulkan Computing Framework.

- **Dozens** instead of thousands of lines of code required
- **Augments** Vulkan interface instead of abstracting it
- **BYOV**: Bring-your-own-Vulkan design to play nice with existing Vulkan applications
- **Non-Vulkan name convention** to disambiguate components

THE LINUX FOUNDATION

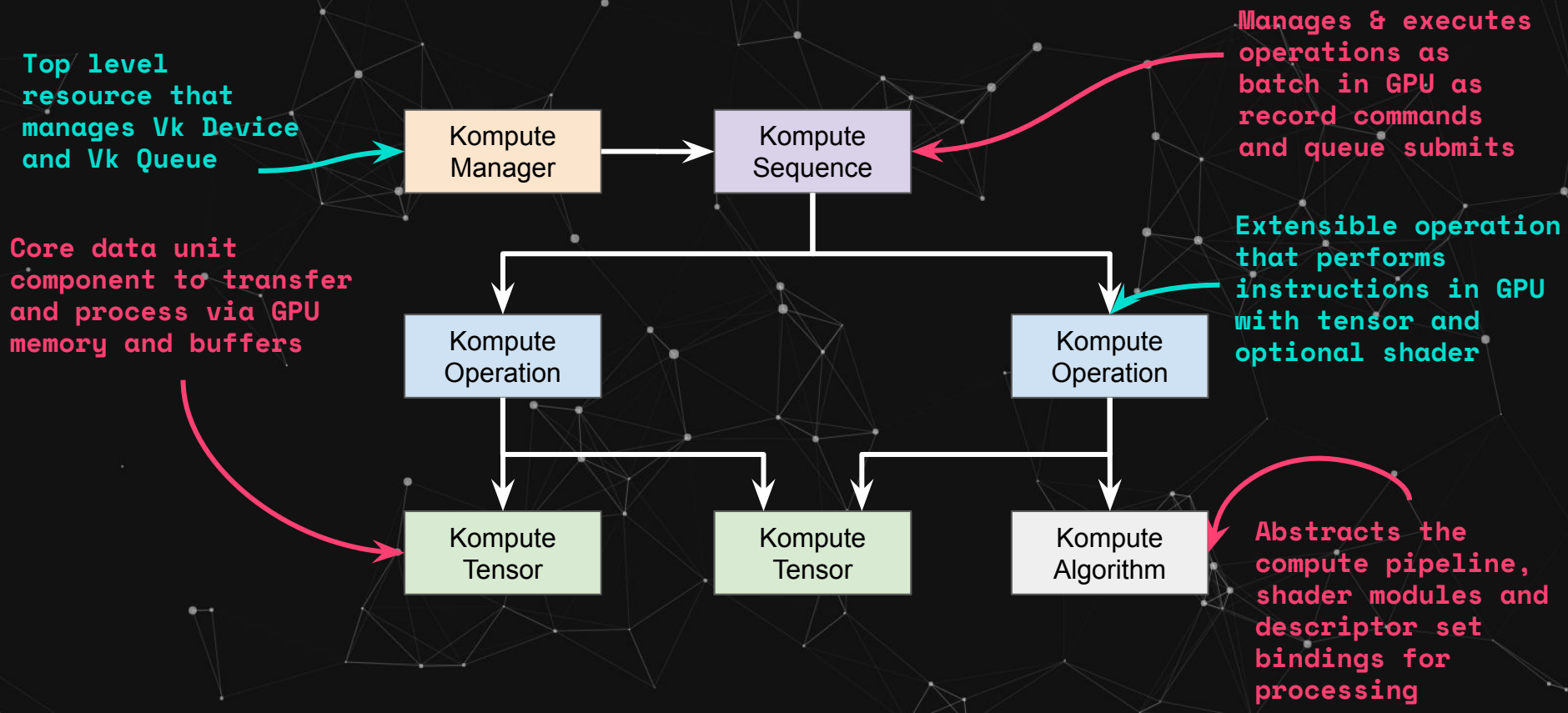
Kompute is part of the Linux Foundation

kompute.cc/overview/community.html

High Level Overview of Features

- C++ Interface
- Python Bindings
- Explicit (GPU/CPU) memory ownership
- Granular access to GPU queues
- Single header file available
- Integration with Mobile Apps
- Integration with Game Engine

Vulkan Kompute: Components



Enter Vulkan Kompute (Simple Sum Example)

```
// 1. Create Kompute Manager
kp::Manager mgr;
```

```
// 2. Create and initialise Kompute Tensors through manager
auto tensorInA = mgr.tensor({ 2., 2., 2. });
auto tensorInB = mgr.tensorT<float>({ 1., 2., 3. });
auto tensorOut = mgr.tensorT<float>({ 0., 0., 0. });

auto params = { tensorInA, tensorInB, tensorOut };
```

```
static std::vector<uint32_t> shader = compileShader(R"(
    #version 450

    layout (local_size_x = 1) in;

    // The input tensors bind index is relative
    layout(binding = 0) buffer bina { float tina[]; };
    layout(binding = 1) buffer binb { float tinb[]; };
    layout(binding = 2) buffer bout { float tout[]; };

    void main() {
        uint index = gl_GlobalInvocationID.x;
        tout[index] = tina[index] * tinb[index];
    }
)");
```

```
// 3. Run operation synchronously
auto algo = mgr.algorithm(params, shader);
```

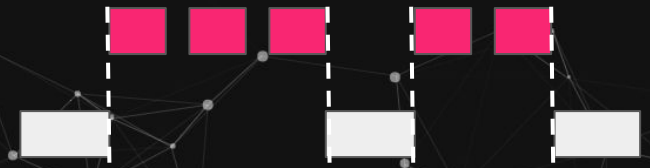
```
// 4. Copy Tensor and execute algorithm
mgr.sequence()
    ->record<kp::OpTensorSyncDevice>(params);
    ->record<kp::OpAlgoDispatch>(algo);
    ->record<kp::OpTensorSyncLocal>(params)
    ->eval();
```

```
// Prints the output which is Output: { 2, 4, 6 }
for (const float& elem : tensorOut->data())
        std::cout << elem << "  ";
```

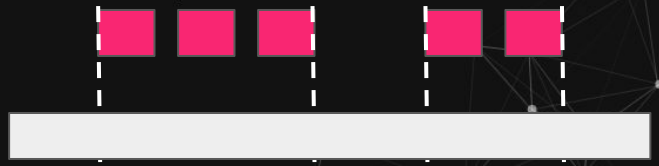
Deeper Optimizations

CPU GPU

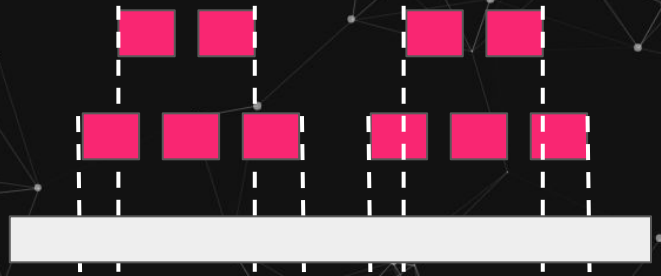
Run a single command/operation in a sequence with manager



Reuse multiple sequences in same Tensors with pre-recorded cmds

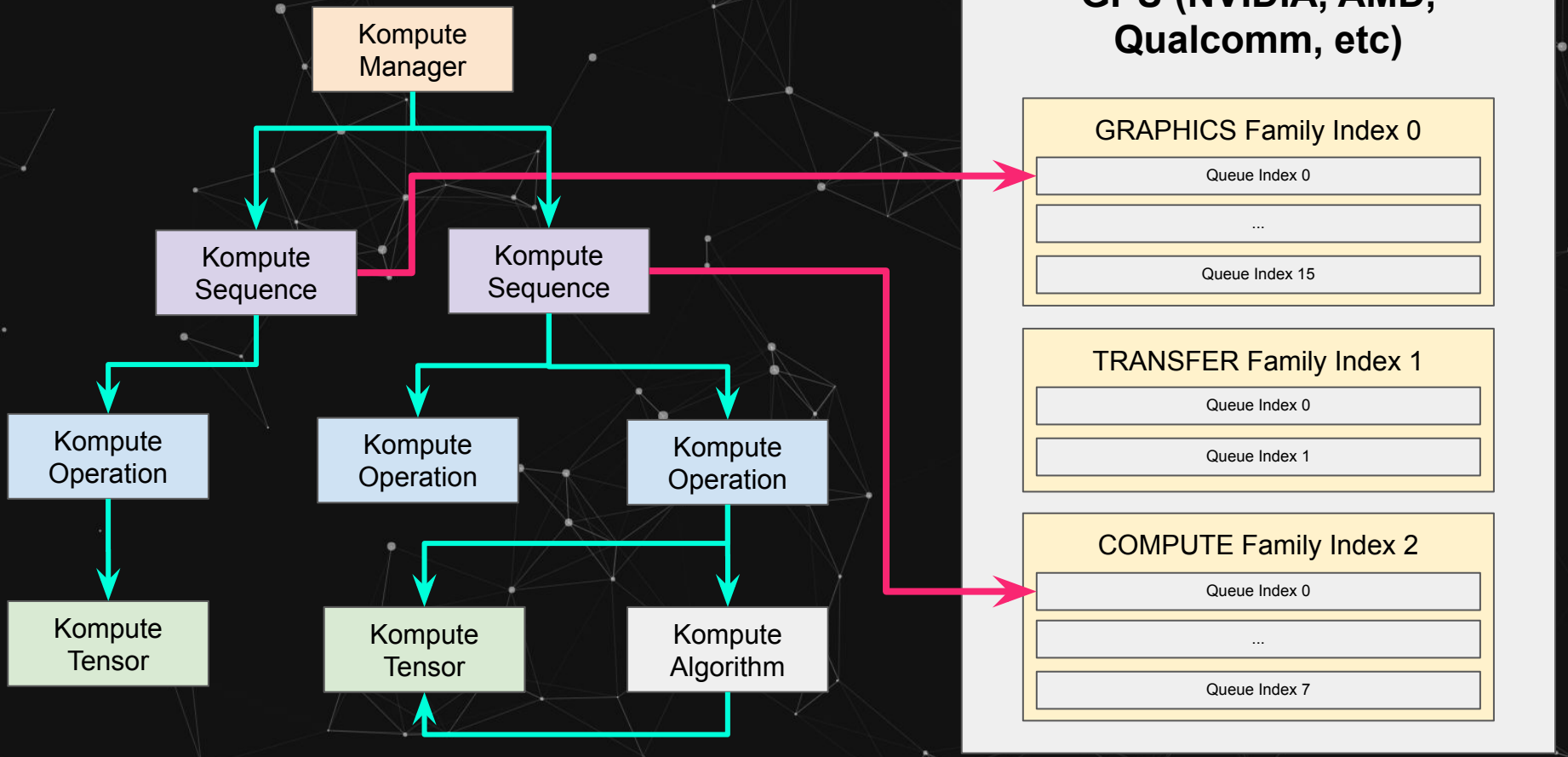


Asynchronous execution of Sequences



Concurrent execution of Sequences across GPU queues

Kompute GPU Optimizations



Enter Vulkan Kompute (Hardware Parallel)

```
// Kompute Manager with custom settings
uint32_t deviceId(1);
std::vector<uint32_t> queues({ 0, 2 });

kp::Manager mgr(deviceId, queues);
```

```
// Create parameters to use for each computation
std::vector<std::shared_ptr<kp::Tensor>> paramsA = { ... };
std::vector<std::shared_ptr<kp::Tensor>> paramsB = { ... };
```

```
// Create seq on relative index
auto sq1 = mgr.sequence(0);
auto sq2 = mgr.sequence(1);
```

```
// Create seq on relative index
auto algo1 = mgr.algorithm(paramsA, shader);
auto algo2 = mgr.algorithm(paramsB, shader);
```

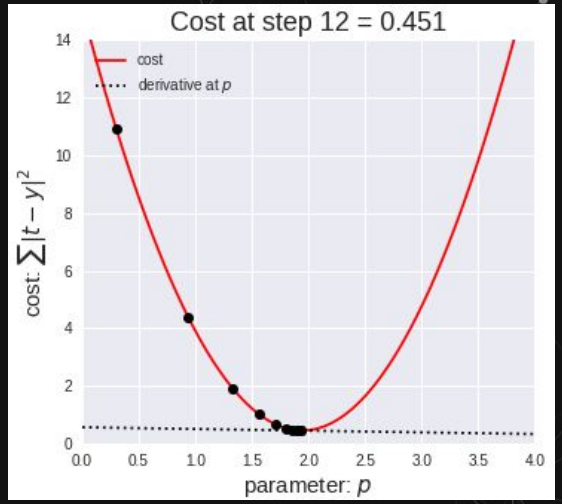
```
sq1->evalAsync<kp::OpaAlgoDispatch>(algo1);

sq2->evalAsync<kp::OpaAlgoDispatch>(algo2);
```

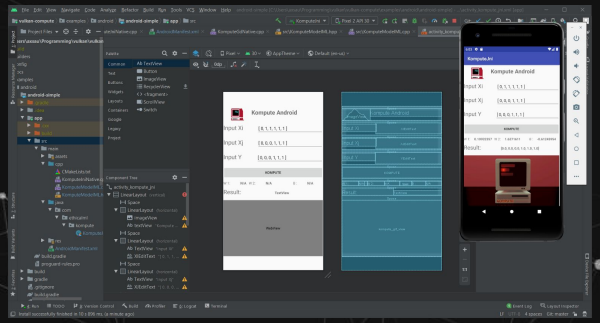
```
sq1->evalAwait();

sq2->evalAwait();
```

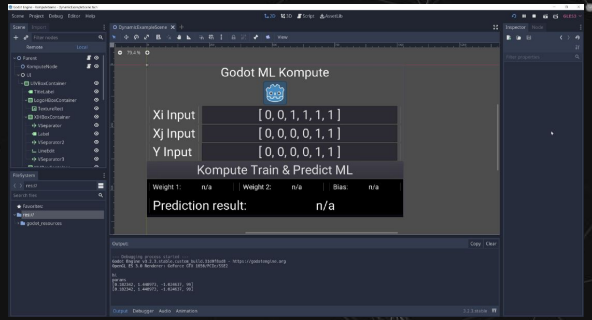
Check out other tutorials



GPU Accelerated Machine Learning
[Blog Post]

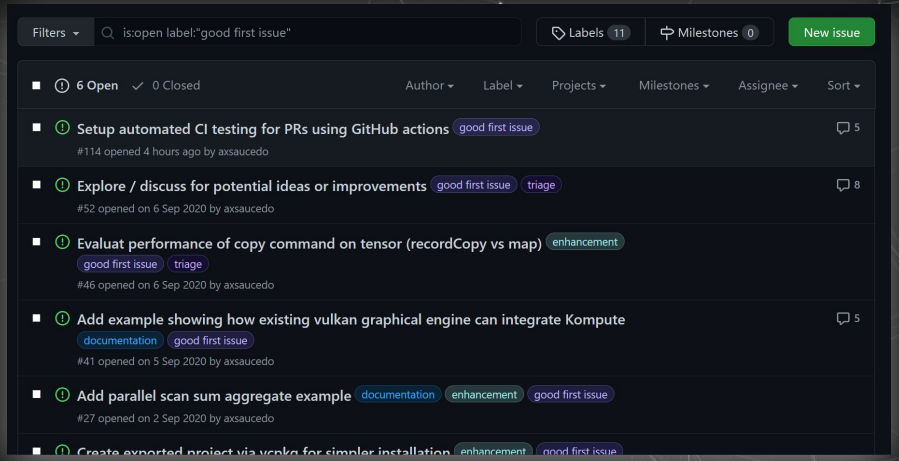


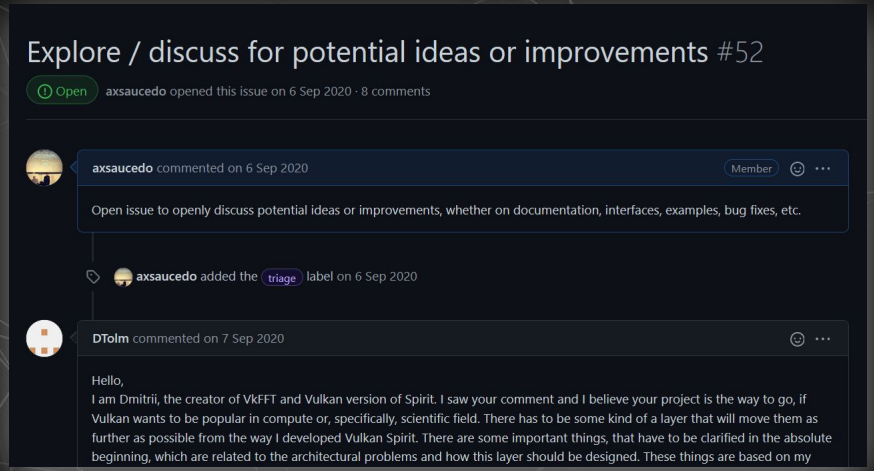
Android App Integration [Blog Post]



Godot Game Engine [Blog Post]

Get Involved!





Pick up one of the good-first-issues

Share thoughts and suggestions via #52

High level Roadmap

Integrate as backend of ML / scientific-computing framework(s)

Create more default kp::Operations to have out of the box commands

---

Examples running Kompute across other platforms and frameworks

CppCon 2022

GPU computing using Vulkan & Kompute for Cross-vendor Graphic Cards (AMD, Qualcomm, NVIDIA & friends)

Alejandro Saucedo

@AxSaucedo