Lab 1

Compute Pipeline

What is a GPU

- **Graphics Processing Unit**
- **Help** the Central Processing Unit (CPU)
- Initially for the calculation of **graphisms**
- More and more for **other things**.

Why a GPU ?

- Optimized for parallel computing:
  - Max cores in a CPU: 128 (very expensive)
  - cores in a GPU: thousands
- The same processing on a large number of data

How to use the GPU?

- 3 **manufacturers** for computers:
  - Intel
  - AMD
  - NVidia
- Many **API**:
  - DirectX: Microsoft
  - Metal: Apple
  - OpenGL/Vulkan: Open

    - CUDA: NVidia Compute API
    - OpenCL: Open Compute API
    - WebGPU

Vulkan

- Modern API: Successor of OpenGL
- Open and “Cross-plateform” (not so much on MacOS)
- Highly explicit API
- Graphics and Compute

Vulkan Logo

WebGPU

- One API to Rule them All
- Designed for the Web and for Standalone Application
- Use preferred API on each platform
- More learnable than Vulkan

WebGPU Logo

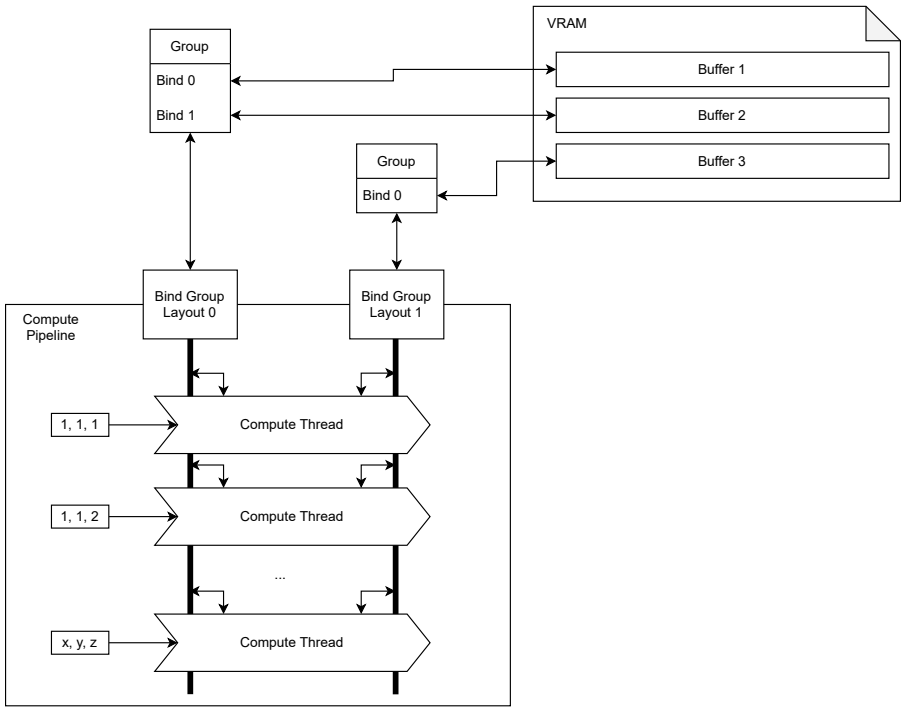
Without GPU: Python Loop

```python
import numpy as np
import time

# Helper class to measure execution time
class Timer:
    def __init__(self, msg: str):
        self.msg = msg

    def __enter__(self):
        self.start = time.perf_counter()

    def __exit__(self, exc_type, exc_value, traceback):
        print(f"{self.msg}: {time.perf_counter() - self.start}"

# Input Data
n = 4194240
numpy_data = np.full(n, 3, dtype=np.int32)

with Timer("for in range loop"):
    res = []
    for x in numpy_data:
        res.append(x * x)
```

Without GPU: numpy

```python
with Timer("numpy operation") as rep:
    res = numpy_data * numpy_data
```

Compute Pipeline



Compute Pipeline

Use WebGPU with python

Terminal

```
pip install wgpu
```

- Get a Logical Device

```
from wgpu import gpu

adapter = gpu.request_adapter_sync(power_preference="high-
device = adapter.request_device_sync()
```

Buffers

- Create Buffer

```
from wgpu import BufferUsage

# input data
buffer0 = device.create_buffer_with_data(data=numpy_data,

# for the output
buffer1 = device.create_buffer(
      size=numpy_data.nbytes,
      usage=BufferUsage.STORAGE | BufferUsage.COPY_SRC,
)
```

- Buffer contains only bytes

  No info on how to interpret them

Bind Group Layout

```
from wgpu import ShaderStage, BufferBindingType

bind_group_layout = device.create_bind_group_layout(
    entries=[
        {
            "binding": 0,
            "visibility": ShaderStage.COMPUTE,
            "buffer": {"type": BufferBindingType.read_only_sto
        },
        {
            "binding": 1,
            "visibility": ShaderStage.COMPUTE,
            "buffer": {"type": BufferBindingType.storage},
        },
    ]
)
```

- Describe the bindings needed in the Bind Group used by the compute operation

Compute Pipeline

```
# Load the source code to execute on the GPU
with open("compute.wgsl") as file:
    shader_source = file.read()

shader_module = device.create_shader_module(code=shader_source)

# A Pipeline may use multiple Bind Group
pipeline_layout = device.create_pipeline_layout(
    bind_group_layouts=[bind_group_layout]
)

pipeline = device.create_compute_pipeline(
    layout=pipeline_layout,
    compute={
        "module": shader_module,
        "entry_point": "main",
    },
)
```

WGSL Language

```
@group(0) @binding(0)
var<storage,read> data0: array<i32>;

@group(0) @binding(1)
var<storage,read_write> data1: array<i32>;

@compute
@workgroup_size(64)
fn main(@builtin(global_invocation_id) index: vec3<u32>) {
    let i: u32 = index.x;
    data1[i] = data0[i] * data0[i];
}
```

- `index` is marked with `@builtin(global_invocation_id)` so it will receive the 3 ints thread ID
- Here we define how to interpret buffer's bytes
- WGSL syntax is close to the Rust syntax

Bind Group

- We must create the Bind Group now

```
bind_group = device.create_bind_group(
    layout=bind_group_layout,
    entries=[
        {
            "binding": 0,
            "resource": {
                "buffer": buffer0,
                "offset": 0,
                "size": buffer0.size,
            },
        },
        {
            "binding": 1,
            "resource": {
                "buffer": buffer1,
                "offset": 0,
                "size": buffer1.size,
            },
        },
    ],
)
```

Run the pipeline

- We must create and send Commands to the GPU

```
from wgpu import GPUCommandEncoder, GPUComputePassEncoder

with Timer("Compute Pipeline"):
    command_encoder: GPUCommandEncoder = device.create_command_

    compute_pass: GPUComputePassEncoder = command_encoder.begi

    compute_pass.set_pipeline(pipeline)
    compute_pass.set_bind_group(0, bind_group)
    compute_pass.dispatch_workgroups(n // 64, 1, 1)
    compute_pass.end()

    device.queue.submit([command_encoder.finish()])
```

Get data back

```
out: memoryview = device.queue.read_buffer(buffer1)
result = np.frombuffer(out.cast("i"), dtype=np.int32)
print(result)
```

Hands-on

Game of Life