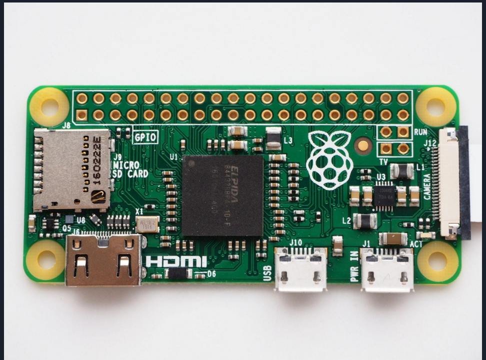
BLAS Libraries for the Raspberry Pi Quad Processing Unit

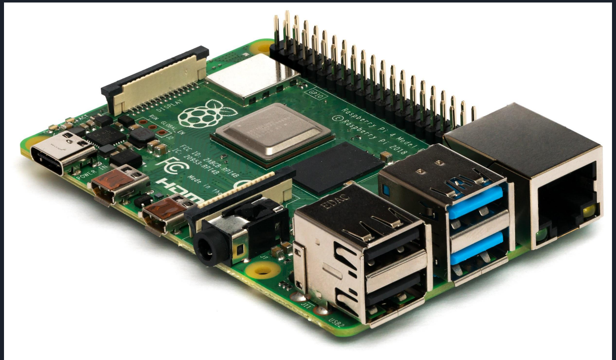
By: Kashyap Panda, Nicole Garcia & Emily Romero

What is a Raspberry Pi?

- The Raspberry Pi is an embedded system which consists of a series of small single-board computers.

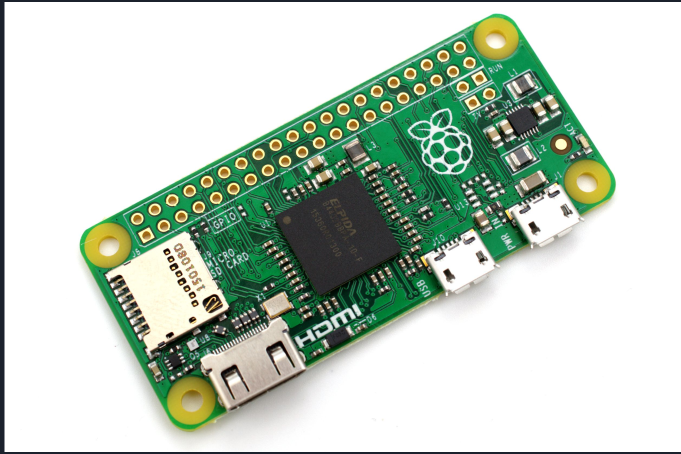
- Due to its programmability , it has a broad number of applications.





Features of the Pi Zero

- Tiny
- Built in Wifi
- Great for small projects
- Not as much computing power
- cost : $5 - $10



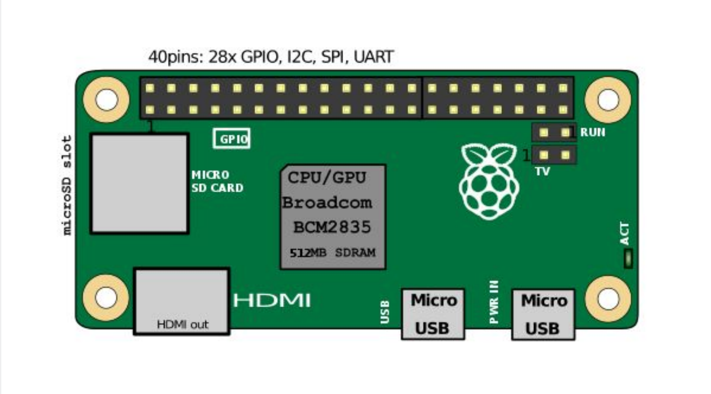
Features of the Pi Four

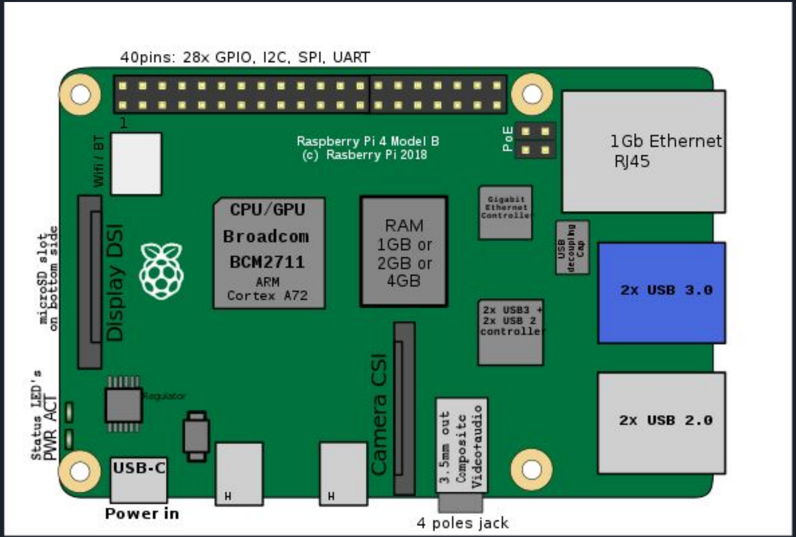
- Four usb ports
- Ethernet Jack
- 4gb of memory
- Built in wifi
- Quad core processor :
  - helps with speed performance
- General Purpose Input/output pins:
  - Used to send and receive electrical signals : meaning we can control things that use electricity to run
- cost : $35+

Hardware Overview

- Each model has common elements:
  - CPU
  - GPU (Videocore 4 or 6)
  - RAM
  - I/O

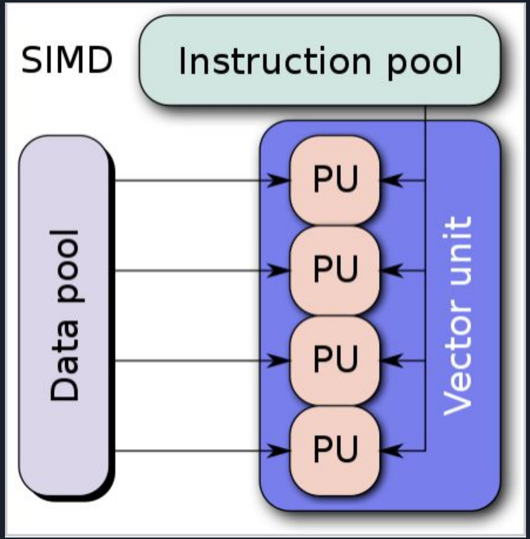
- The main focus is on the Videocore GPU





What is a QPU?

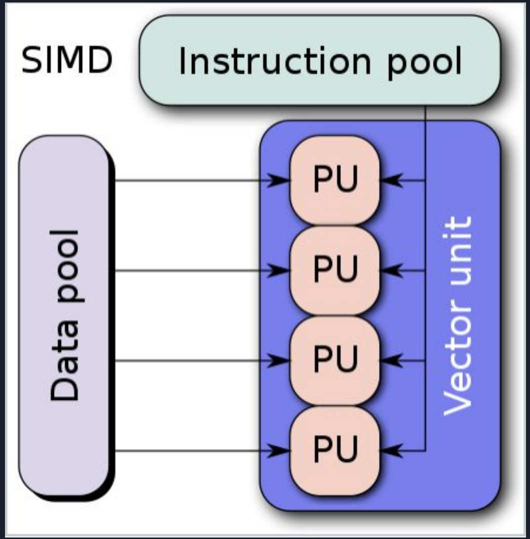
- The main part of the Videocore Unit
- Stands for **Q**uad **P**rocessing **U**nit
- 16-way 32-bit SIMD processor
  - **S**ingle **I**nstruction, **M**ultiple **D**ata
  - Simultaneously performs the same operation on multiple data points
    - In this case, on 16 data points

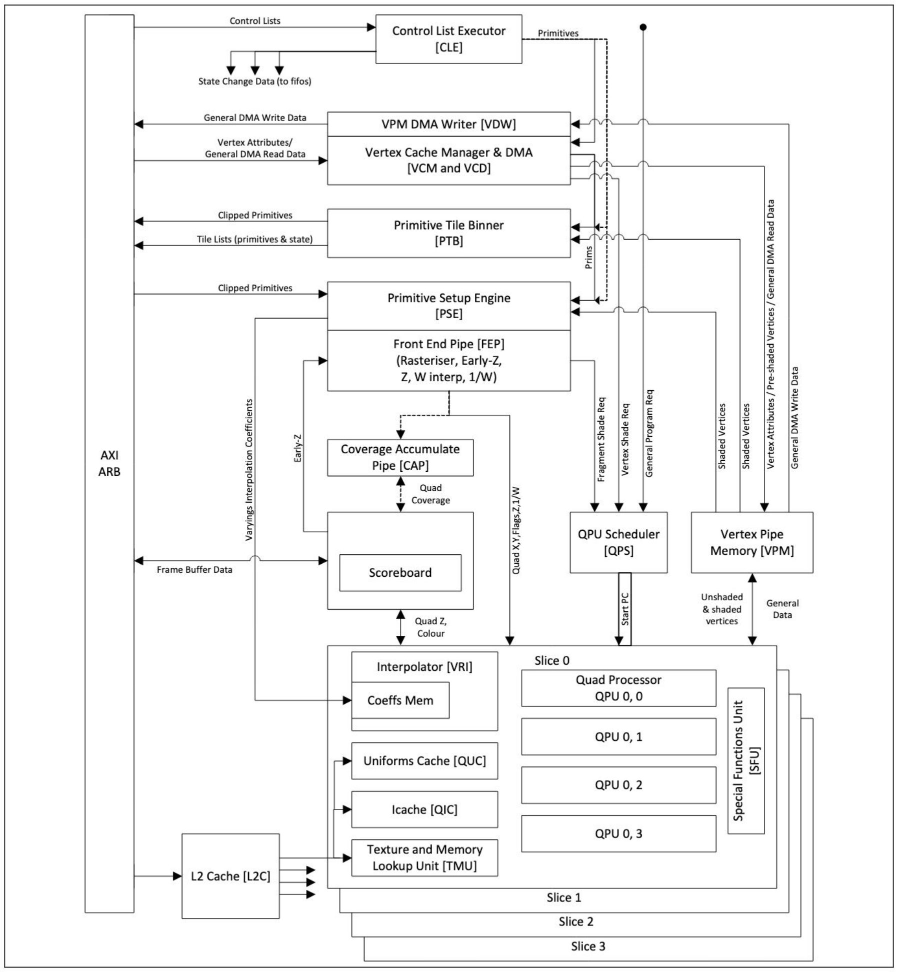


What is a QPU?

- Breaks the 16 points into 4 **quads**
  - 4 sets of 4 points
  - Where the **Q** in QPU comes from

- Processes 1 quad per clock cycle
  - Processes everything in 4 clock cycles

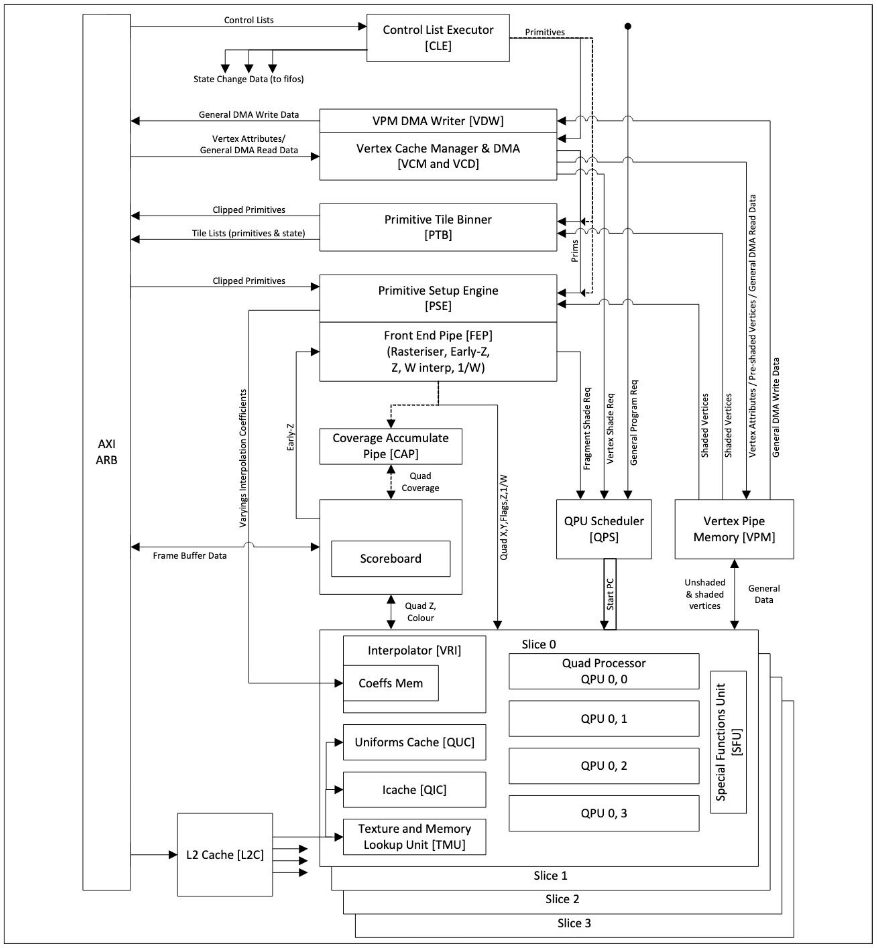




AXI ARB
Control Lists
Control List Executor [CLE]
Primitives
State Change Data (to fifos)
General DMA Write Data
VPM DMA Writer [VDW]
Vertex Attributes/ General DMA Read Data
Vertex Cache Manager & DMA [VCM and VCD]
Clipped Primitives
Tile Lists (primitives & state)
Primitive Tile Binner [PTB]
Prims
Clipped Primitives
Primitive Setup Engine [PSE]
Front End Pipe [FEP] (Rasteriser, Early-Z, Z, W interp, 1/W)
Varyings Interpolation Coefficients
Early-Z
Coverage Accumulate Pipe [CAP]
Quad Coverage
Scoreboard
Frame Buffer Data
Quad Z, Colour
Quad X,Y,Flags,Z,1/W
Fragment Shade Req
Vertex Shade Req
General Program Req
Shaded Vertices
Shaded Vertices
Vertex Attributes / Pre-shaded Vertices / General DMA Read Data
General DMA Write Data
QPU Scheduler [QPS]
Vertex Pipe Memory [VPM]
Start PC
Unshaded & shaded vertices
General Data
Slice 0
Interpolator [VRI]
Coeffs Mem
Quad Processor QPU 0, 0
QPU 0, 1
QPU 0, 2
QPU 0, 3
Special Functions Unit [SFU]
Uniforms Cache [QUC]
Icache [QIC]
Texture and Memory Lookup Unit [TMU]
L2 Cache [L2C]
Slice 1
Slice 2
Slice 3

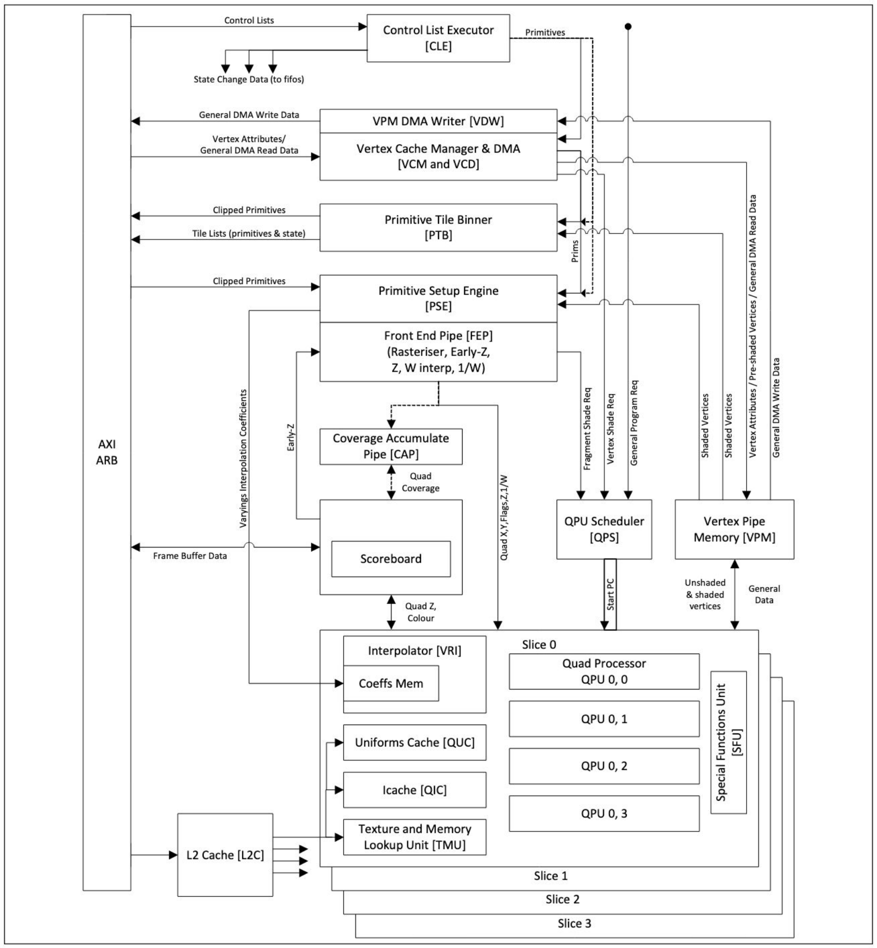
QPU Architecture

- QPUs are organized into groups/slices
- 12 QPUs at 250 MHz on Videocore 4 (Pi Zero)
- 12 QPUs at 300 MHz on Videocore 4 (Pi 3)
- 8 QPUs at 500 MHz on Videocore 6 (Pi 4)



QPU Architecture

- Each QPU has access to other components to enable functionality
  - Registers
  - Uniforms
  - Texture and Memory Lookup Unit (TMU)
  - Special Functions Unit (SFU)
  - And more!



Theoretical Outputs

- FLOP/S
    - Floating-point Operations Per Second
    - 1 GFLOP/S = 1000000000 FLOP/S = 10^9 FLOP/S

- Videocore 4 (Pi Zero)
    - 250 MHz * 3 slices * 4 QPUs/slice * 4 cycles* 2 ops/cycle = 24 GFLOP/S

- Videocore 4 (Pi 3)
    - 300 MHz * 3 slices * 4 QPUs/slice * 4 cycles* 2 ops/cycle = 28.8 GFLOP/S

- Videocore 6
    - 500 MHz * 2 slices * 4 QPUs/slice * 4 cycles* 2 ops/cycle = 32 GFLOP/S

Videocore 4 vs 6, General Comparison

	Raspberry Pi Zero	Raspberry Pi 3	Raspberry Pi 4
GPU	Videocore 4	Videocore 4	Videocore 6
Clock Speed	250 MHz	300 MHz	500 MHz
# of QPUs	12	12	8
# of slices	3	3	2
Theoretical GFLOP/S	24	28.8	32

Current Goals

- While the QPU has processing power, most programs and routines built for the Raspberry Pi don’t utilize it

- Optimizing code by using the QPU would provide a massive speedup throughout the entire device

- Use optimized code to compare the cost and energy efficiency of the different Raspberry Pi GPUs

How to write code for the QPU

- The QPU is programmed using assembly language
- Contains the typical add, subtract, or, etc.
- The Videocore 4 and Videocore 6 have different instruction sets

Instruction	***opcode***	***Description***
nop	0	No operation
fadd	1	Floating point add
fsub	2	Floating point subtract
fmin	3	Floating point min
fmax	4	Floating point max
fminabs	5	Floating point min of absolute values
fmaxabs	6	Floating point max of absolute values
ftoi	7	Floating point to signed integer
itof	8	Signed integer to floating point
–	9–11	Reserved
add	12	Integer add
sub	13	Integer subtract
shr	14	Integer shift right
asr	15	Integer arithmetic shift right
ror	16	Integer rotate right
shl	17	Integer shift left
min	18	Integer min
max	19	Integer max
and	20	Bitwise AND

How to write code for the QPU

- There are libraries available that implement the QPU assembly language as part of other languages
  - Videocore 4
    - py-videocore
    - VC4CL
  - Videocore 6
    - py-videocore6
  - Both
    - V3DLib

- Each library has its own set of benefits and drawbacks

About the Libraries

- py-videocore and py-videocore6

- Python libraries used for programming on raspberry pi boards

- Allow us to communicate with the V3D GPU hardware (a driver used by the raspberry pi)

- Directly maps QPU Python function to the QPU assembly instructions
    - E.g. there is an add function written in Python that directly calls the QPU add instruction

- We have been using these libraries to run tests and analyze the performance rate of the GPU vs the QPU on the raspberry pi systems

About the Libraries

- V3DLib
  - A C++ library used for creating programs to run on the VideoCore GPU on all Raspberry Pi boards

  - Doesn’t directly map functions to assembly like py-videocore, creates a high-level C++ API that is converted to QPU assembly
    - Runs the program on the CPU and offloads to the QPU at runtime

  - Compiles on both VideoCore IV and VideoCore VI

  - We have been using this library to run tests and analyze the performance rate of one QPU vs eight QPU’s on the raspberry pi systems

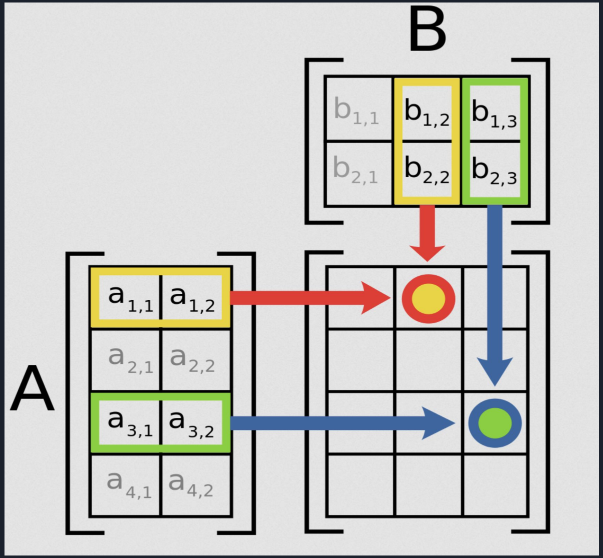
About the Libraries

- VC4CL
  - Runs the OpenCL language directly on the Videocore 4 GPU
    - Converts OpenCL code to assembly
    - Not supported for Videocore 6
  - Has the benefit of using OpenCL, which is used for a variety of GPUs and devices
  - Not usable for benchmarks, has significant compatibility issues when running OpenCL code on the Videocore 4

Benchmarks

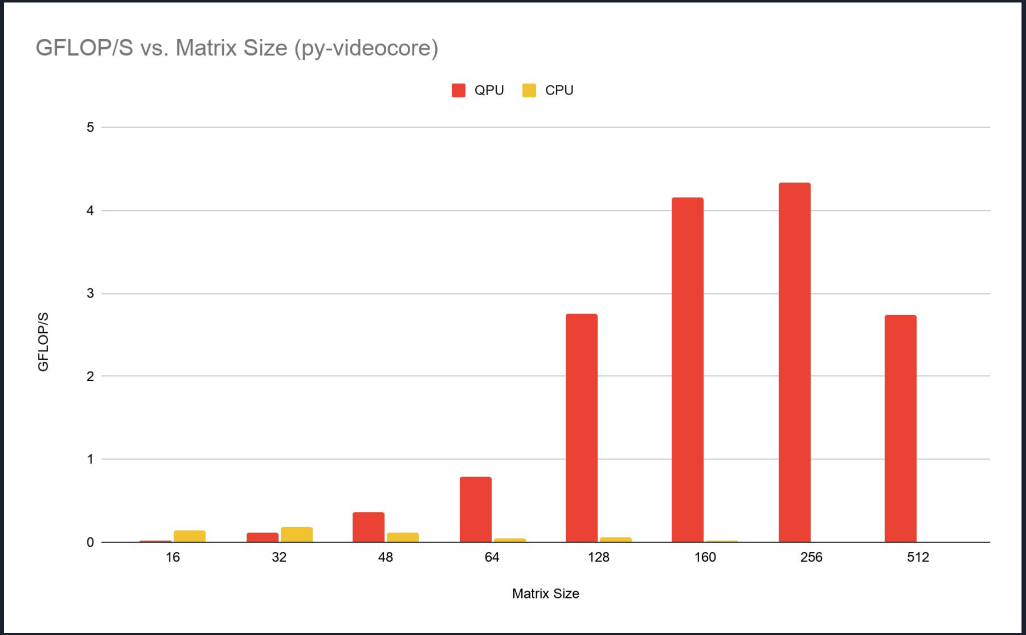
- Multiply matrices directly on the QPU

- Measure GFLOP/S
  - Calculate the number of operations
  - Measure the time to completion
  - GFLOP/S = (operations / time) * 10^9



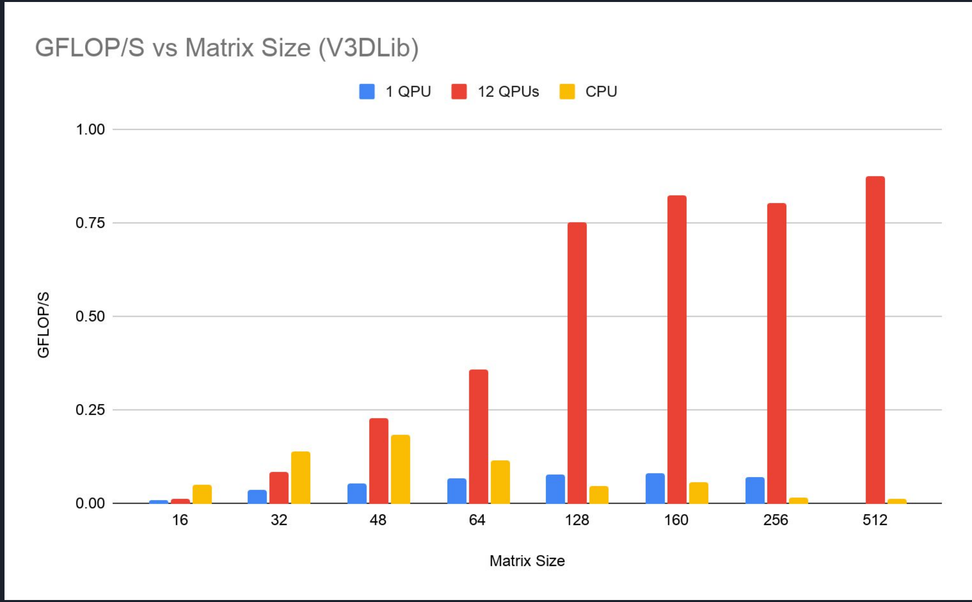
Results - Videocore 4

Matrix Size	12 QPUs (GFLOP/S)	CPU (GFLOP/S)
16	0.0164	0.0497
32	0.1098	0.1392
48	0.3598	0.1837
64	0.7891	0.1135
128	2.7519	0.0451
160	4.1552	0.0571
256	4.336	0.0154
512	2.7411	0.0111



Results - Videocore 4

Matrix Size	1 QPU (GFLOP/S)	12 QPUs (GFLOP/S)	CPU (GFLOP/S)
16	0.0087	0.0126	0.0497
32	0.0356	0.0856	0.1392
48	0.0536	0.2294	0.1837
64	0.0657	0.3575	0.1135
128	0.0765	0.7522	0.0451
160	0.0802	0.8244	0.0571
256	0.0712	0.8036	0.0154
512	ERROR	0.8762	0.0111



Results - Videocore 4

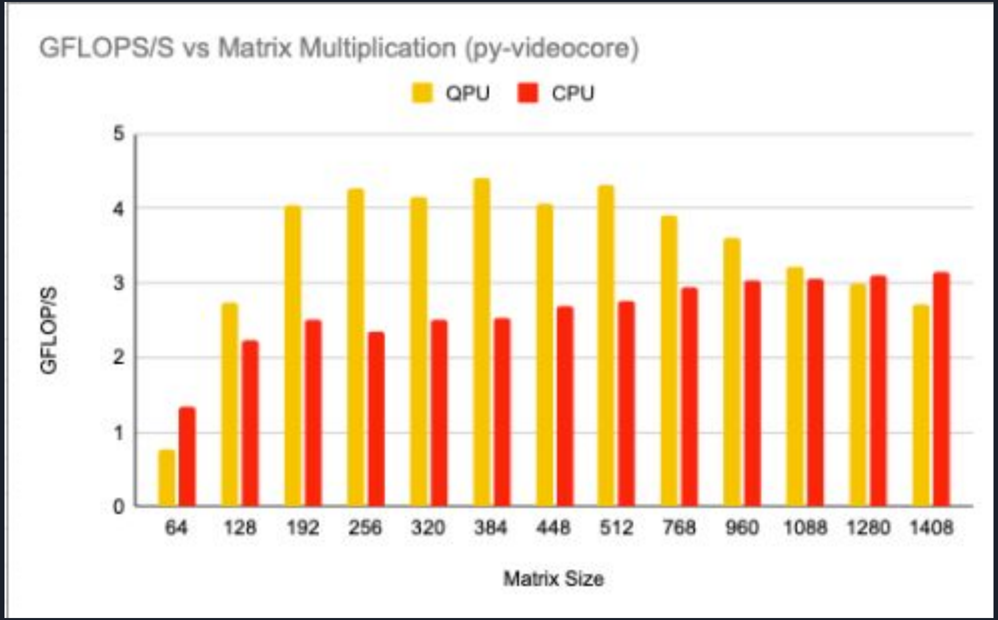
- Despite their speed differences, both libraries outperformed the CPU on matrix multiplication, especially with large
    - Py-videocore ran matrix multiplication much faster than V3Dlib
    - Due to less overhead and more direct interfacing with the QPU

- Different matrix sizes affect GFLOP/S
    - Achieved 8.7 GFLOP/S with 96x363 and 363x3072 matrices

- The Raspberry Pi Zero dangerously heats up when running this
    - It is a good idea to repeat this with proper power measurement tools

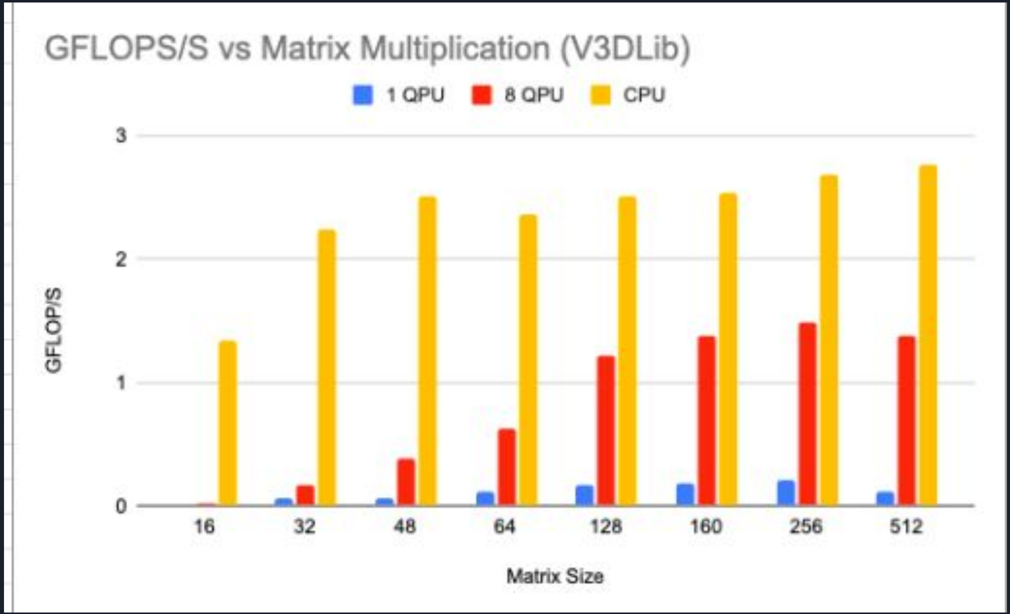
Results - Videocore 6

Matrix Size	QPUs (GFLOP/S)	CPU (GFLOP/S)
64	0.7798	1.336
128	2.746	2.243
192	4.045	2.501
256	4.271	2.354
320	4.152	2.501
384	4.399	2.539
448	4.06	2.685
512	4.322	2.766



Results - Videocore 6

Matrix Size	1 QPU (GFLOP/S)	8 QPUs (GFLOP/S)	CPU (GFLOP/S)
16	0.004053	0.024646	1.336
32	0.059568	0.161280	2.243
48	0.057149	0.38400	2.501
64	0.117160	0.626737	2.354
128	0.169393	1.221971	2.501
160	0.183535	1.376901	2.539
256	0.205640	1.479911	2.685
512	0.109250	1.38213	2.766



Results - Videocore 6

- Using the Py-videocore library our QPU runs faster than the CPU with a matrix 128x128 - 1088 x 1088

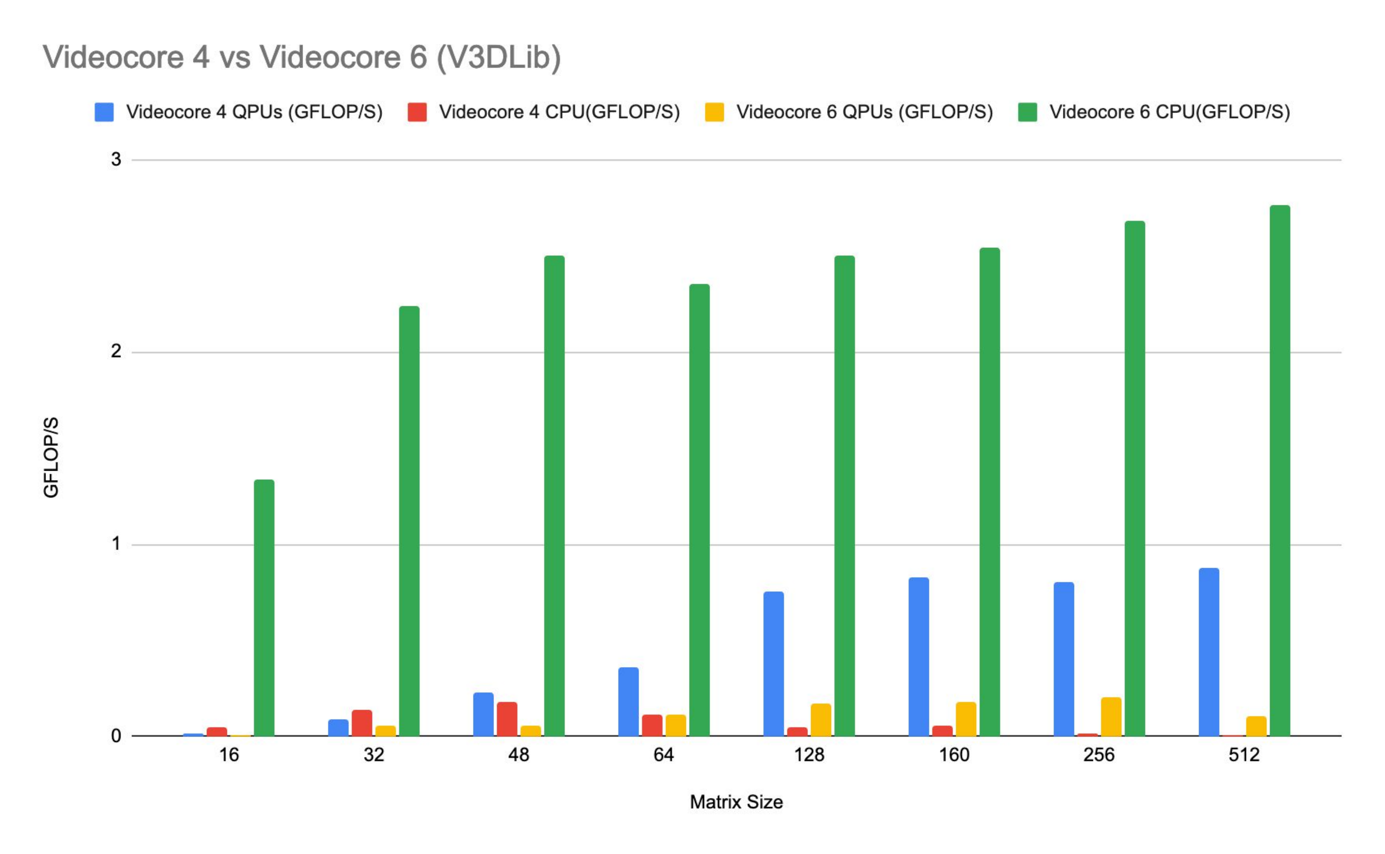
- Using the V3D Lib library the CPU outperforms the QPU by a great amount

- Findings : The Py videocore 6 library ran matrix multiplication significantly faster than the VD3Lib library

Videocore 4 vs 6, Benchmark Comparison

Matrix Size	Videocore 4 QPUs (GFLOP/S)	Videocore 4 CPU(GFLOP/S)	Videocore 6 QPUs (GFLOP/S)	Videocore 6 CPU(GFLOP/S)
16	0.0126	0.0497	0.004053	1.336
32	0.0856	0.1392	0.059568	2.243
48	0.2294	0.1837	0.057149	2.501
64	0.3575	0.1135	0.117160	2.354
128	0.7522	0.0451	0.169393	2.501
160	0.8244	0.0571	0.183535	2.539
256	0.8036	0.0154	0.205640	2.685
512	0.8762	0.0111	0.109250	2.766

Videocore 4 vs 6, Benchmark Comparison



Videocore 4 vs 6, Benchmark Comparison

- Depending on the library, the Videocore 4 sometimes outperforms the Videocore 6

- py-videocore and py-videocore6 always outperform V3DLib when executing code on the QPU

- Likely due to the difference between assembly implementations
    - Faster to directly map assembly to functions, rather than making a high-level API

Future Goals

- We plan to further expand our understanding by continuing to run tests and analyzing the performance of the system
    - Overclock the Raspberry Pi 4

- Implement BLAS
    - **B**asic **L**inear **A**lgebra **S**ubprograms
    - Common linear algebra operations like dot products, matrix multiplication, etc.

- Libraries that implement BLAS already exist for both Videocore GPUs
    - qmkl and qmkl6 (which use py-videocore and py-videocore6 underneath)
    - Current libraries are either incompatible with all models or have an incomplete featureset

Future Goals

- Improve performance and compatibility with models by making modifications to the BLAS libraries

- This will help with analysis of more complex libraries which will also increase the capabilities of the Raspberry Pi

- Be able to use the BLAS libraries on the QPU to accelerate machine learning frameworks, like PyTorch

- Use the optimized frameworks and BLAS libraries to measure power consumption

QUESTIONS?