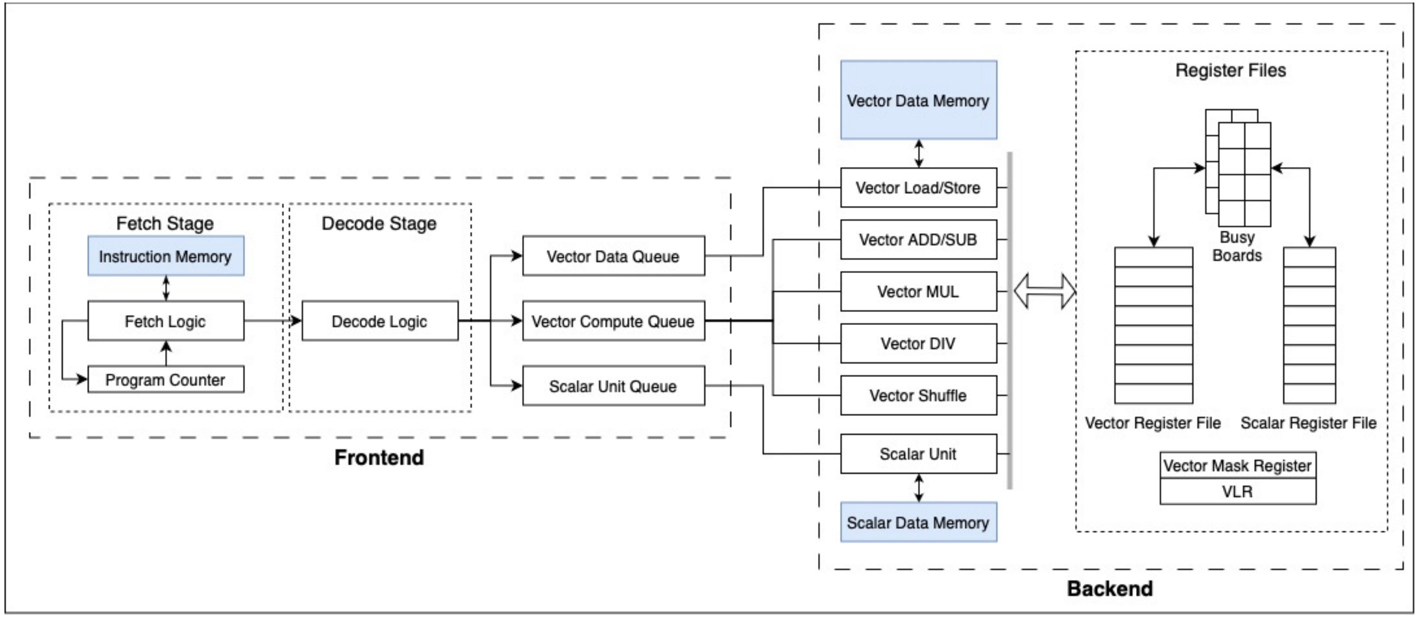
NYU

Vector Processor

Performance Simulator

Akshath Mahajan, Rugved Mhatre



Overview of Architecture

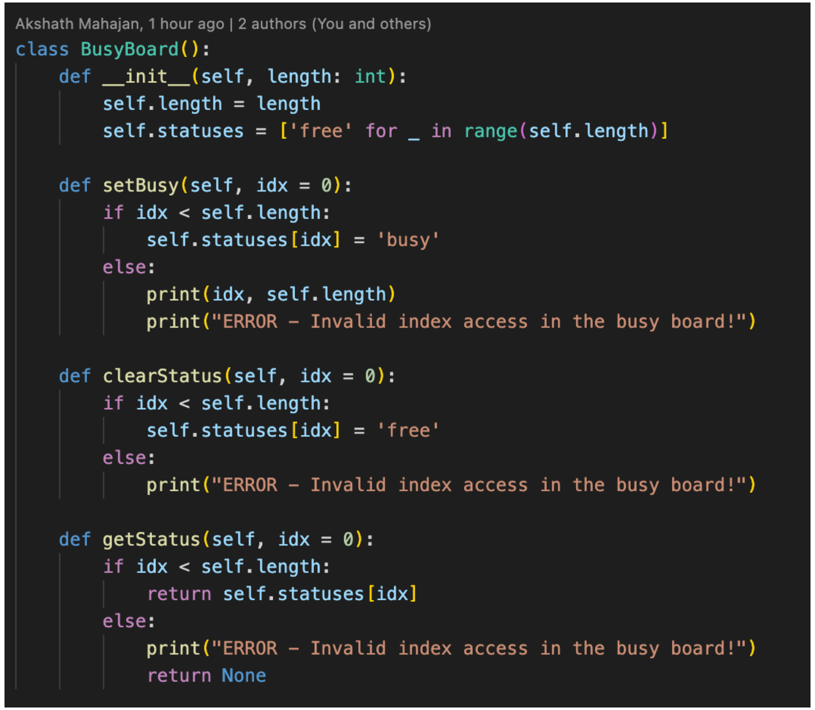
NYU

Code Overview

- Execute
- Decode
- Dispatch To Queues
- Pop From Queues
- Fetch New Instruction

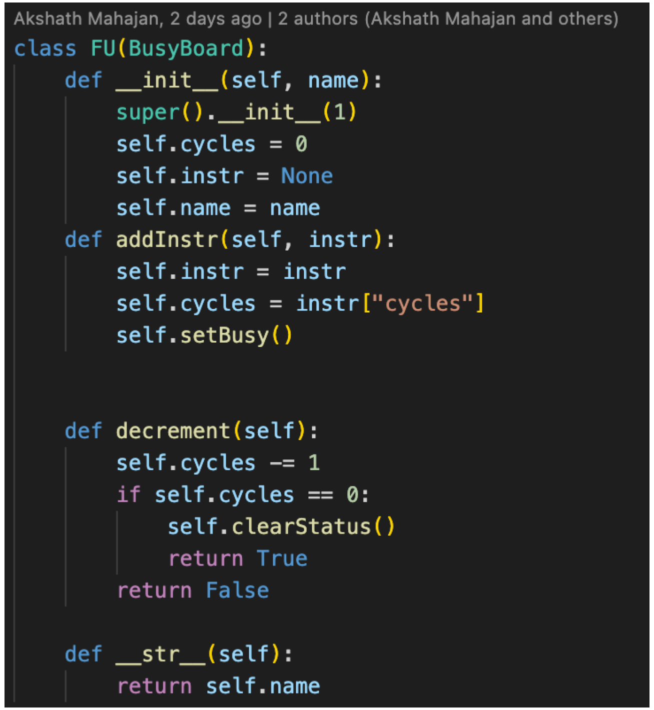
PART 01

Code Overview



Akshath Mahajan, 1 hour ago | 2 authors (You and others)

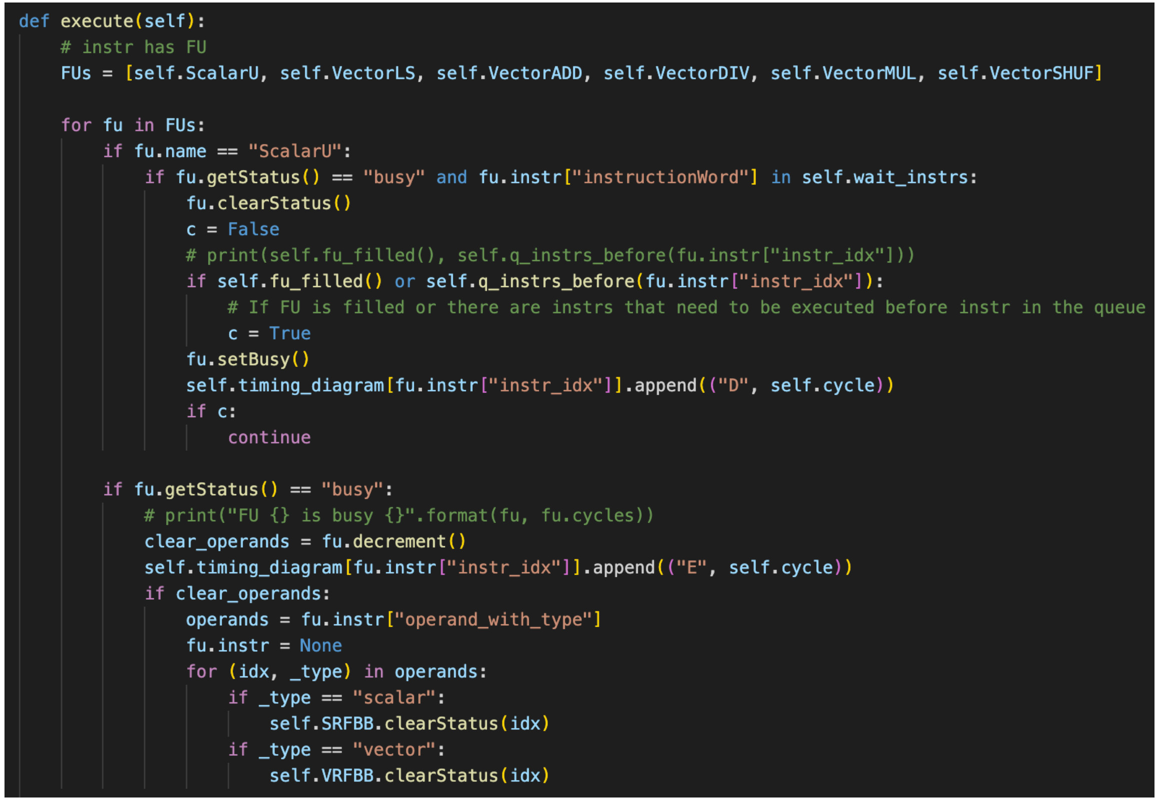
```python
class BusyBoard():
    def __init__(self, length: int):
        self.length = length
        self.statuses = ['free' for _ in range(self.length)]

    def setBusy(self, idx = 0):
        if idx < self.length:
            self.statuses[idx] = 'busy'
        else:
            print(idx, self.length)
            print("ERROR - Invalid index access in the busy board!")

    def clearStatus(self, idx = 0):
        if idx < self.length:
            self.statuses[idx] = 'free'
        else:
            print("ERROR - Invalid index access in the busy board!")

    def getStatus(self, idx = 0):
        if idx < self.length:
            return self.statuses[idx]
        else:
            print("ERROR - Invalid index access in the busy board!")
            return None
```



Akshath Mahajan, 2 days ago | 2 authors (Akshath Mahajan and others)

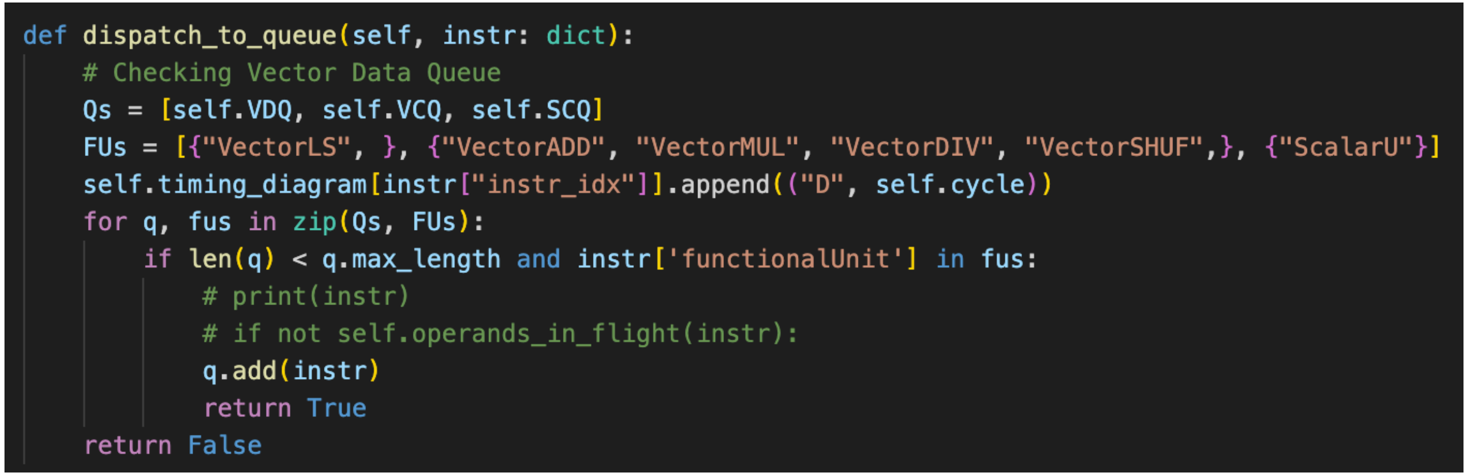
```python
class FU(BusyBoard):
    def __init__(self, name):
        super().__init__(1)
        self.cycles = 0
        self.instr = None
        self.name = name
    def addInstr(self, instr):
        self.instr = instr
        self.cycles = instr["cycles"]
        self.setBusy()


    def decrement(self):
        self.cycles -= 1
        if self.cycles == 0:
            self.clearStatus()
            return True
        return False

    def __str__(self):
        return self.name
```

Code Overview



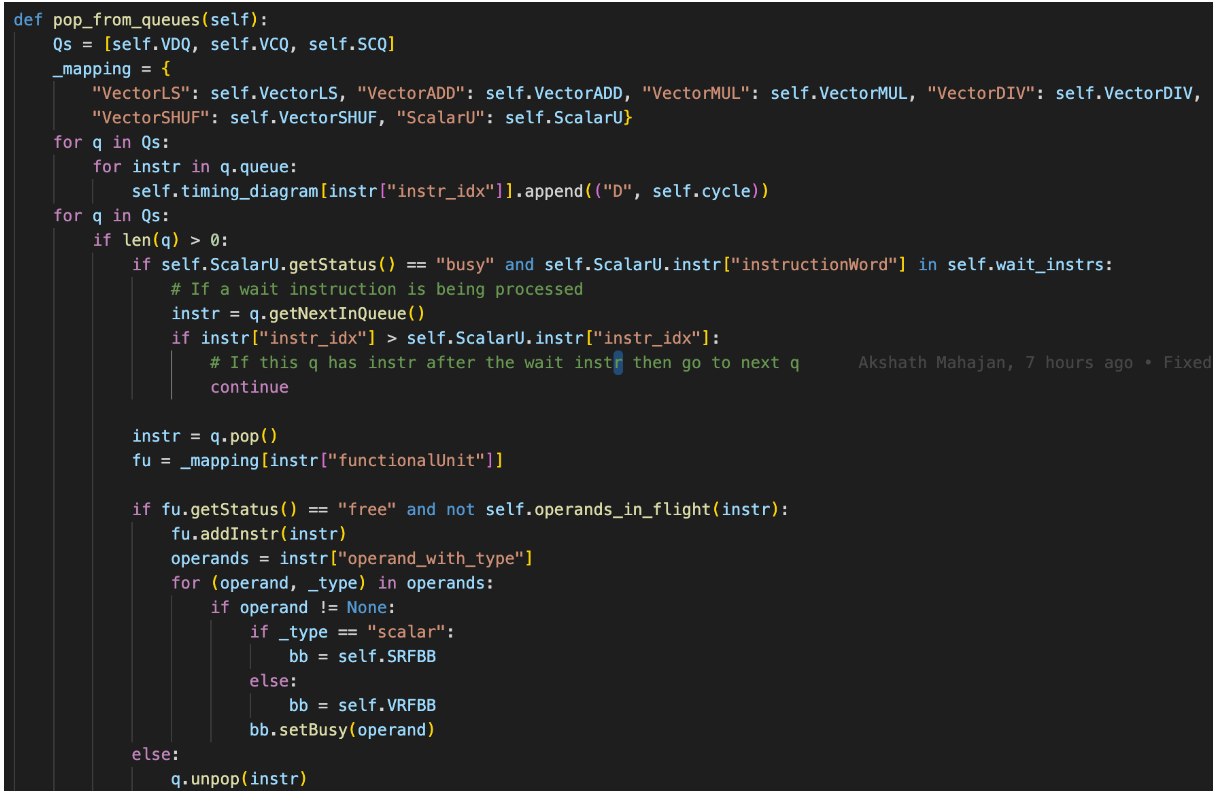
```python
def execute(self):
    # instr has FU
    FUs = [self.ScalarU, self.VectorLS, self.VectorADD, self.VectorDIV, self.VectorMUL, self.VectorSHUF]

    for fu in FUs:
        if fu.name == "ScalarU":
            if fu.getStatus() == "busy" and fu.instr["instructionWord"] in self.wait_instrs:
                fu.clearStatus()
                c = False
                # print(self.fu_filled(), self.q_instrs_before(fu.instr["instr_idx"]))
                if self.fu_filled() or self.q_instrs_before(fu.instr["instr_idx"]):
                    # If FU is filled or there are instrs that need to be executed before instr in the queue
                    c = True
                fu.setBusy()
                self.timing_diagram[fu.instr["instr_idx"]].append(("D", self.cycle))
                if c:
                    continue

        if fu.getStatus() == "busy":
            # print("FU {} is busy {}".format(fu, fu.cycles))
            clear_operands = fu.decrement()
            self.timing_diagram[fu.instr["instr_idx"]].append(("E", self.cycle))
            if clear_operands:
                operands = fu.instr["operand_with_type"]
                fu.instr = None
                for (idx, _type) in operands:
                    if _type == "scalar":
                        self.SRFBB.clearStatus(idx)
                    if _type == "vector":
                        self.VRFBB.clearStatus(idx)
```

Code Overview



```python
def dispatch_to_queue(self, instr: dict):
    # Checking Vector Data Queue
    Qs = [self.VDQ, self.VCQ, self.SCQ]
    FUs = [{"VectorLS", }, {"VectorADD", "VectorMUL", "VectorDIV", "VectorSHUF",}, {"ScalarU"}]
    self.timing_diagram[instr["instr_idx"]].append(("D", self.cycle))
    for q, fus in zip(Qs, FUs):
        if len(q) < q.max_length and instr['functionalUnit'] in fus:
            # print(instr)
            # if not self.operands_in_flight(instr):
            q.add(instr)
            return True
    return False
```

Code Overview



```python
def pop_from_queues(self):
    Qs = [self.VDQ, self.VCQ, self.SCQ]
    _mapping = {
        "VectorLS": self.VectorLS, "VectorADD": self.VectorADD, "VectorMUL": self.VectorMUL, "VectorDIV": self.VectorDIV,
        "VectorSHUF": self.VectorSHUF, "ScalarU": self.ScalarU}
    for q in Qs:
        for instr in q.queue:
            self.timing_diagram[instr["instr_idx"]].append(("D", self.cycle))
    for q in Qs:
        if len(q) > 0:
            if self.ScalarU.getStatus() == "busy" and self.ScalarU.instr["instructionWord"] in self.wait_instrs:
                # If a wait instruction is being processed
                instr = q.getNextInQueue()
                if instr["instr_idx"] > self.ScalarU.instr["instr_idx"]:
                    # If this q has instr after the wait instr then go to next q      Akshath Mahajan, 7 hours ago • Fixed
                    continue

            instr = q.pop()
            fu = _mapping[instr["functionalUnit"]]

            if fu.getStatus() == "free" and not self.operands_in_flight(instr):
                fu.addInstr(instr)
                operands = instr["operand_with_type"]
                for (operand, _type) in operands:
                    if operand != None:
                        if _type == "scalar":
                            bb = self.SRFBB
                        else:
                            bb = self.VRFBB
                        bb.setBusy(operand)
            else:
                q.unpop(instr)
```

PART 02

Testing

Base Configuration

- 1 Scalar Functional Unit
- 1 Vector Load/Store Functional Unit
- 1 Vector Add/Subtract Functional Unit
- 1 Vector Multiply Functional Unit
- 1 Vector Divide Functional Unit
- 1 Vector Shuffle Functional Unit

Dispatch Queue Configuration

- Vector Data Queue Depth - 4
- Vector Compute Queue Depth - 4
- Scalar Compute Queue Depth - 4

Vector Data Memory Configuration

- Vector Data Memory Banks - 16
- Vector Data Memory Bank Busy Time - 2
- Vector Load/Store Pipeline Depth - 11

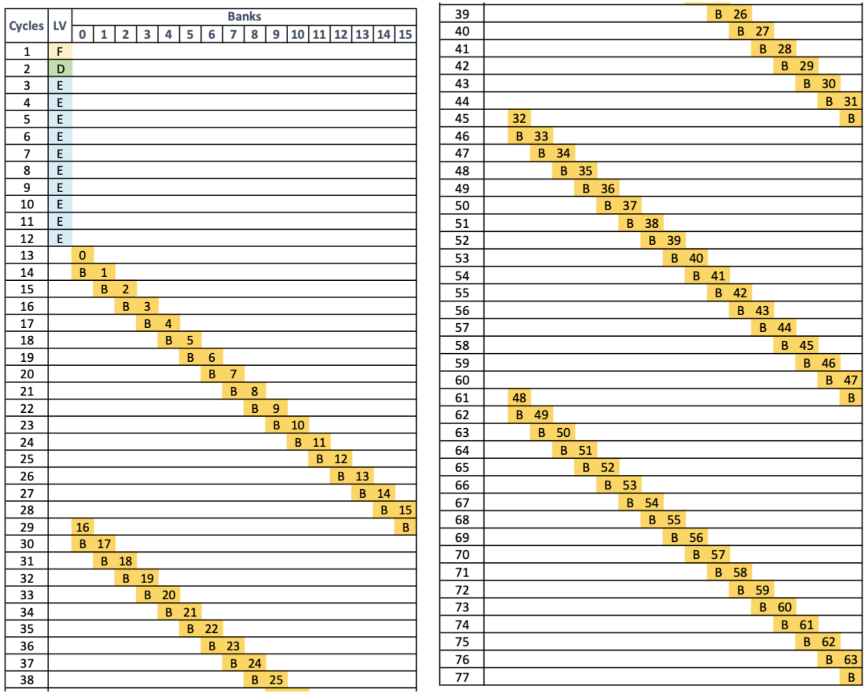
Vector Compute Configuration

- Vector Lanes - 4
- Vector Add/Sub Pipeline Depth - 2
- Vector Multiply Pipeline Depth - 12
- Vector Divide Pipeline Depth - 8
- Vector Shuffle Pipeline Depth - 5

Cycles	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45
LV	F	D	E	E	E	E	E	E	E	E	E	E	B	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
HALT		F	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D

46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	
D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	E

Simple Load Vector Program

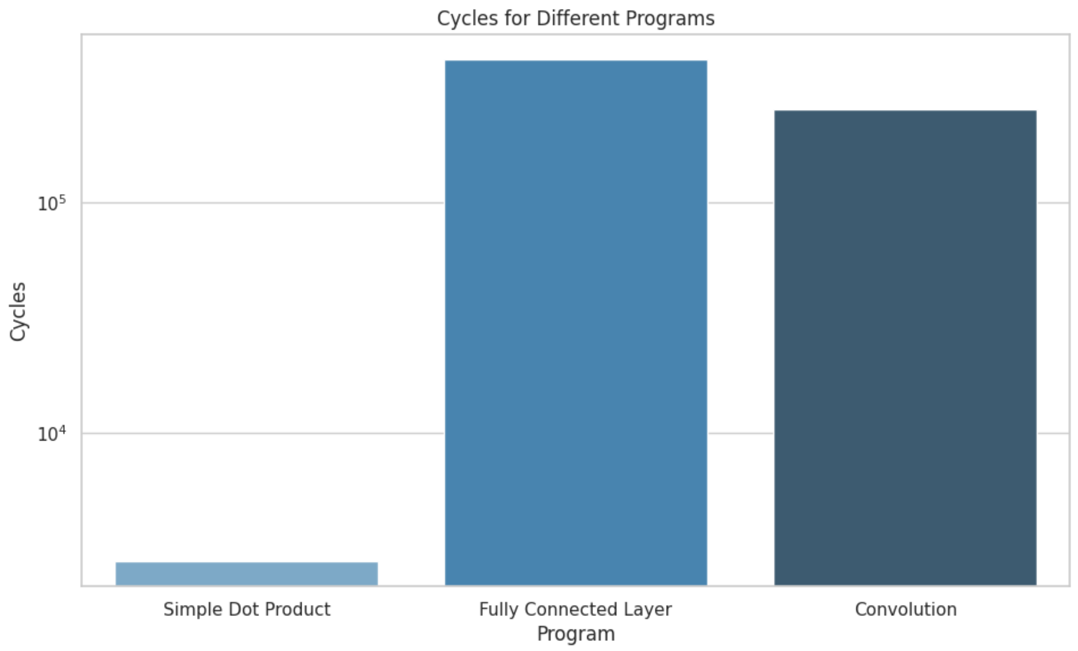


Cycles	LV	Banks															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	F																
2	D																
3	E																
4	E																
5	E																
6	E																
7	E																
8	E																
9	E																
10	E																
11	E																
12	E																
13		0															
14		B	1														
15			B	2													
16				B	3												
17					B	4											
18						B	5										
19							B	6									
20								B	7								
21									B	8							
22										B	9						
23											B	10					
24												B	11				
25													B	12			
26														B	13		
27															B	14	
28																B	15
29		16															B
30		B	17														
31			B	18													
32				B	19												
33					B	20											
34						B	21										
35							B	22									
36								B	23								
37									B	24							
38										B	25						
39											B	26					
40												B	27				
41													B	28			
42														B	29		
43															B	30	
44																B	31
45		32															B
46		B	33														
47			B	34													
48				B	35												
49					B	36											
50						B	37										
51							B	38									
52								B	39								
53									B	40							
54										B	41						
55											B	42					
56												B	43				
57													B	44			
58														B	45		
59															B	46	
60																B	47
61		48															B
62		B	49														
63			B	50													
64				B	51												
65					B	52											
66						B	53										
67							B	54									
68								B	55								
69									B	56							
70										B	57						
71											B	58					
72												B	59				
73													B	60			
74														B	61		
75															B	62	
76																B	63
77																	B

Simple Load Vector Program - Memory Bank View

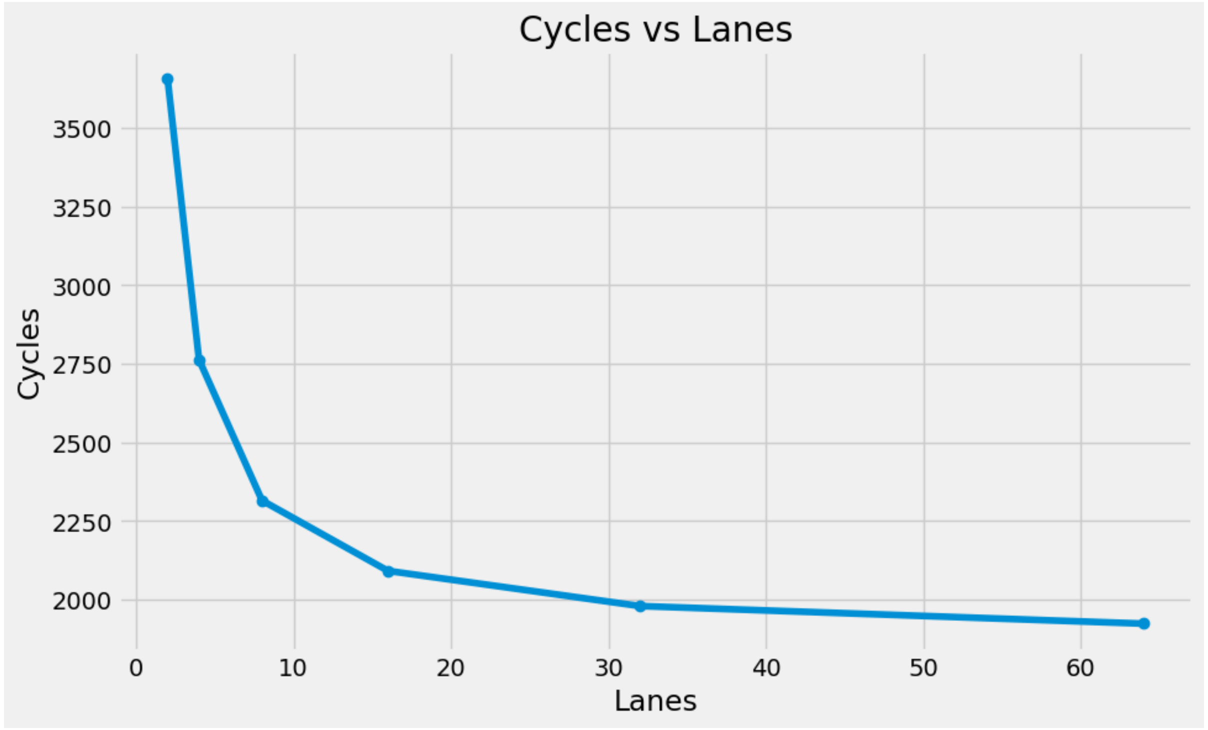
PART 03

Results

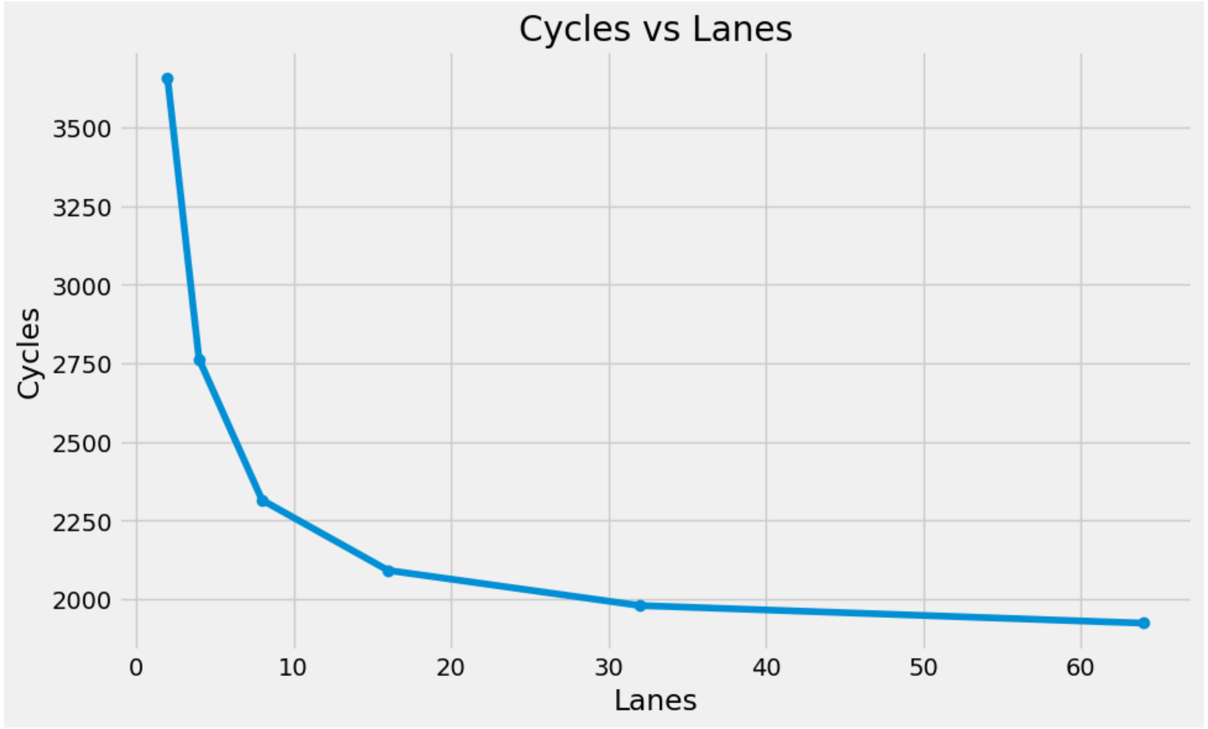


Base Configuration

NYU

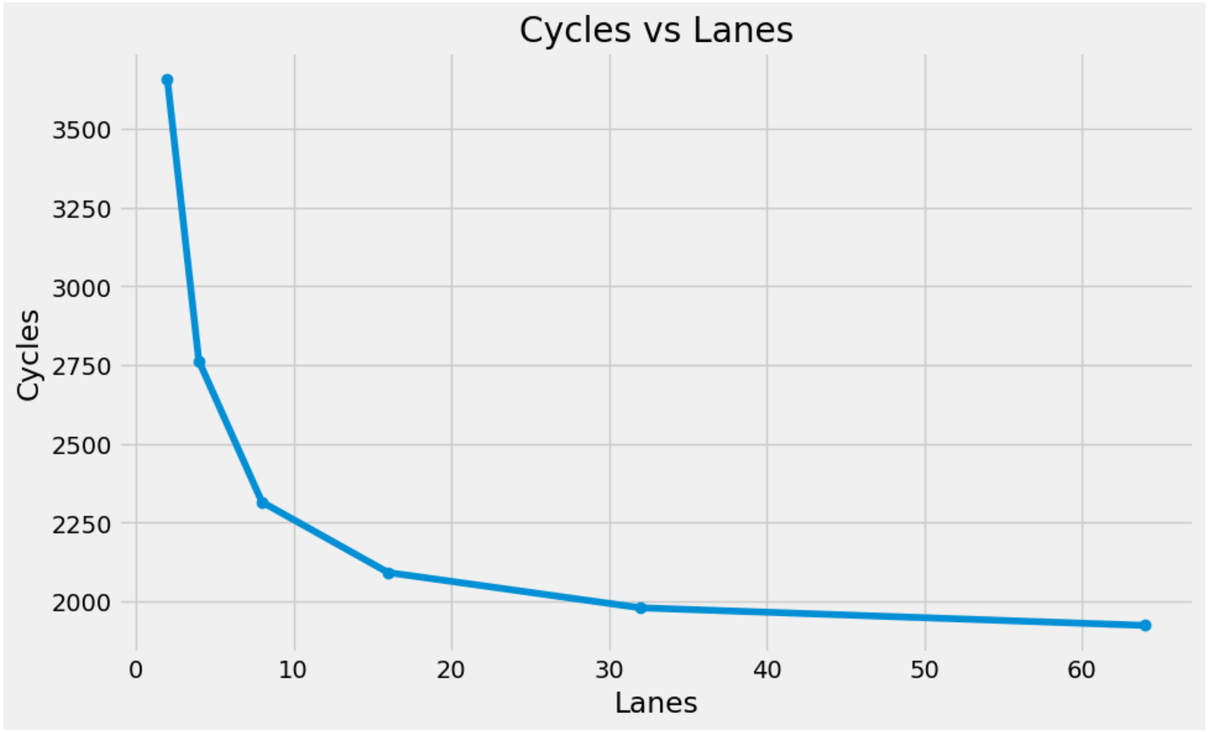


Impact of Number of Lanes (Simple Dot Product)



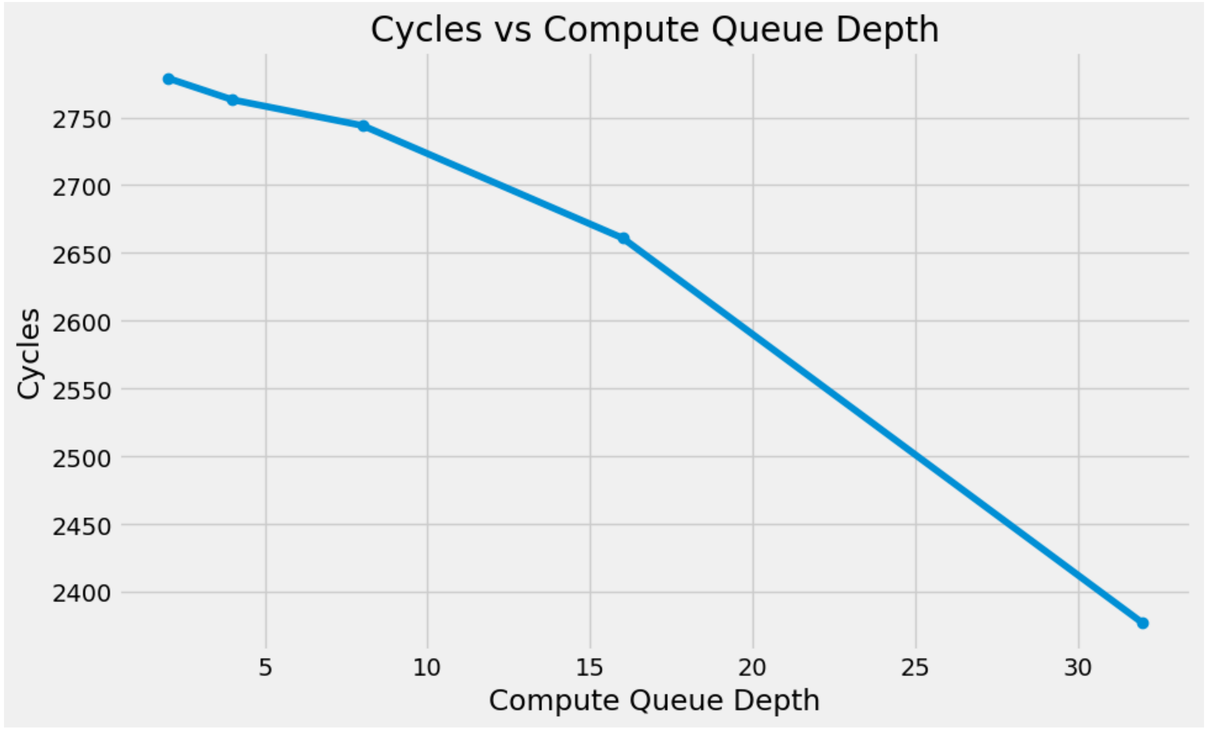
Impact of Number of Lanes (Fully Connected Layer)

NYU

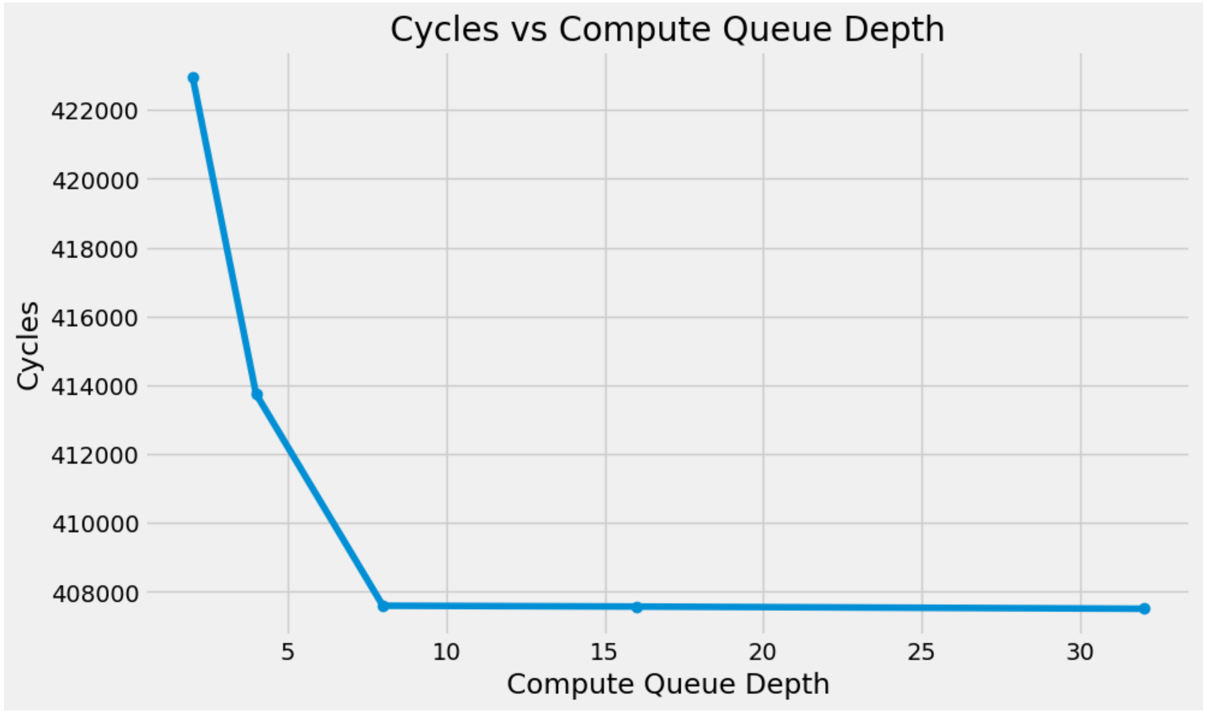


Impact of Number of Lanes (Convolution)

NYU

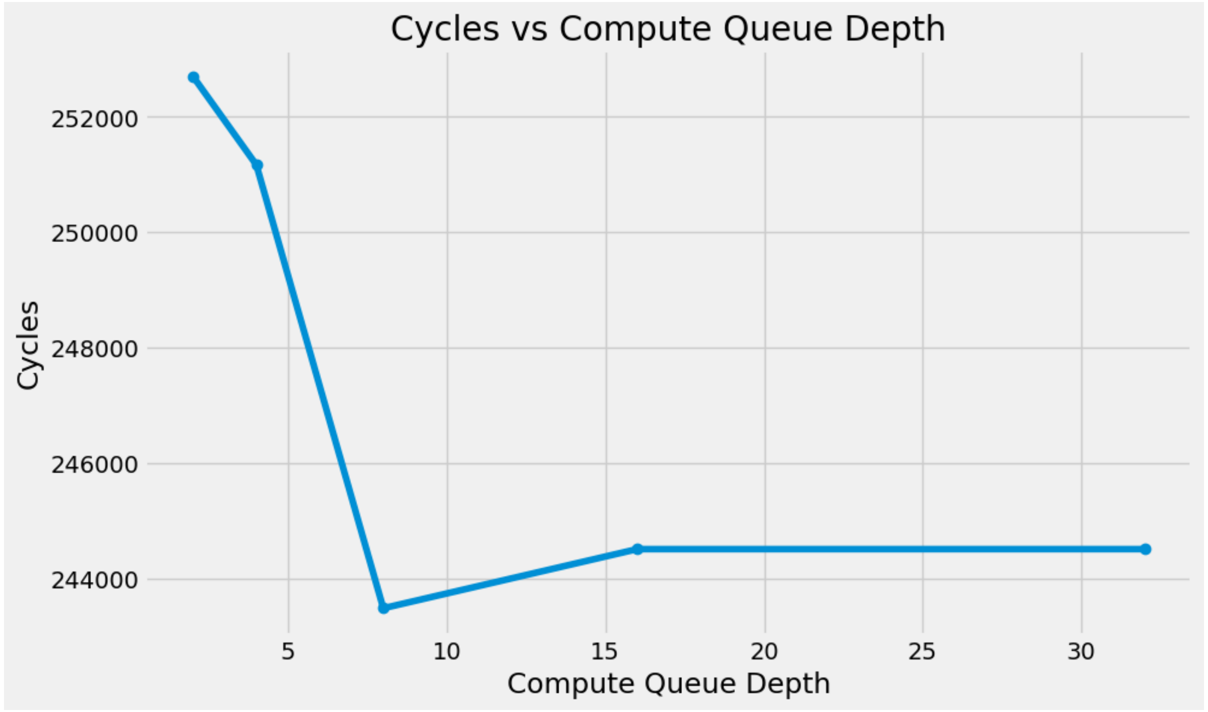


Impact of VCQ Depth (Simple Dot Product)



Impact of VCQ Depth (Fully Connected Layer)

NYU



Impact of VCQ Depth (Convolution)

NYU

Impact of VDQ Depth

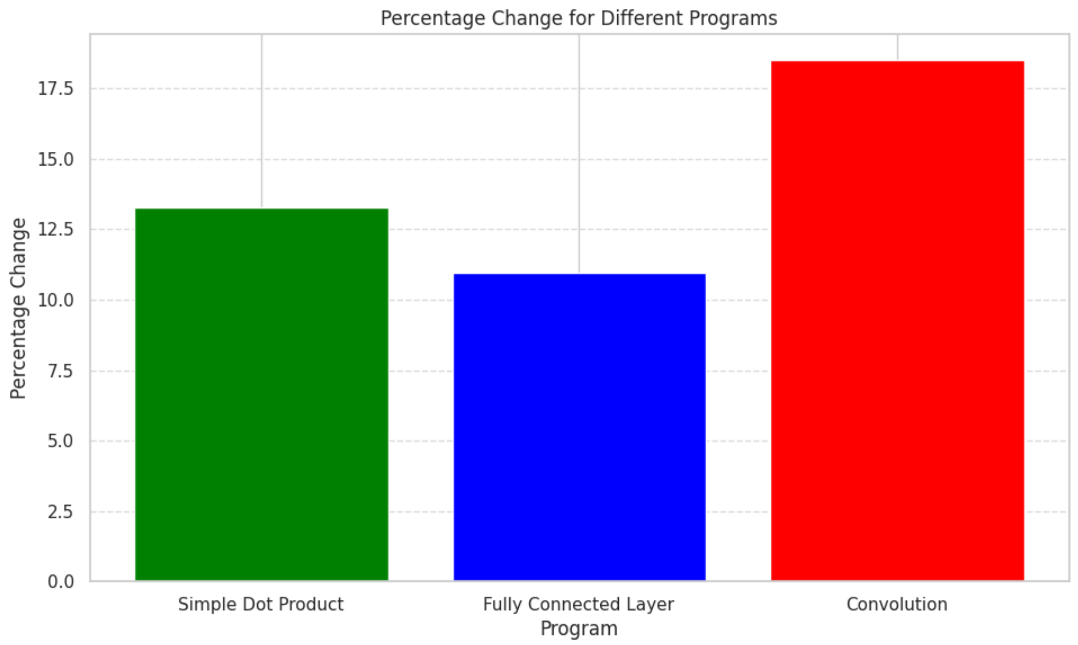
- VDQ Depths doesn't change the number of cycles for depth of 2, 4, 8, 16, 32
- Since we have only one Vector Load Store Unit

Impact of Prime Banks

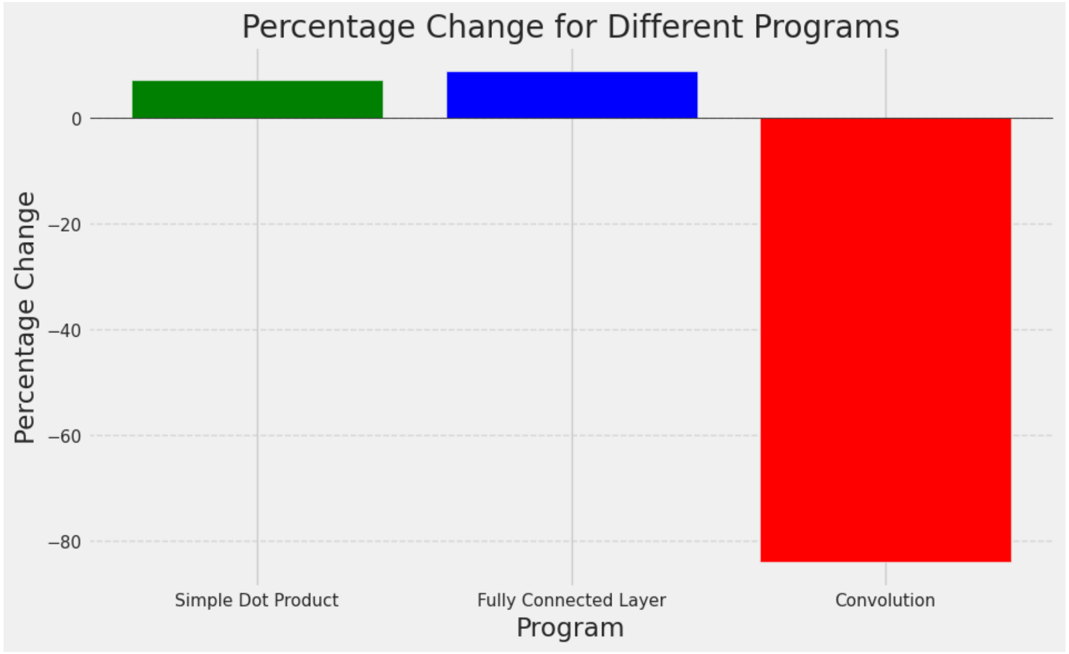
vdNum	Dot Product	Fully Connected	Convolution
2	2763	413747	541469
3	2763	**284723**	**251165**
4	2763	413747	251165
8	2763	413747	251165
16	2763	413747	251165
17	2763	**284723**	**251165**
19	2763	**284723**	**251165**
29	2763	**284723**	**251165**
32	2763	413747	251165
64	2763	413747	251165

PART 04

Optimization



Increased VRF Read Ports



Increased Vector Length (VLR = 128)

PART 05

Conclusion

Conclusion

- Lanes matter the most
- Compute Queues Depth of 8 is the sweet spot
- Data Queue Depth doesn’t matter at all, as we have only one Vector Load Store Unit
- Fully Connected is memory bound, prime banks helps.
- Convolution is compute bound.

Conclusion

- Increasing the Read Ports to the VRFs helps in the execution of more FUs in parallel.
- A longer vector length, doesn't translate well into high performance

Questions?

NYU