CS 758: Advanced Topics in Computer Architecture

Lecture #6: GPU Microarchitecture 3 – Pipeline Operand Stage

Professor Matthew D. Sinclair

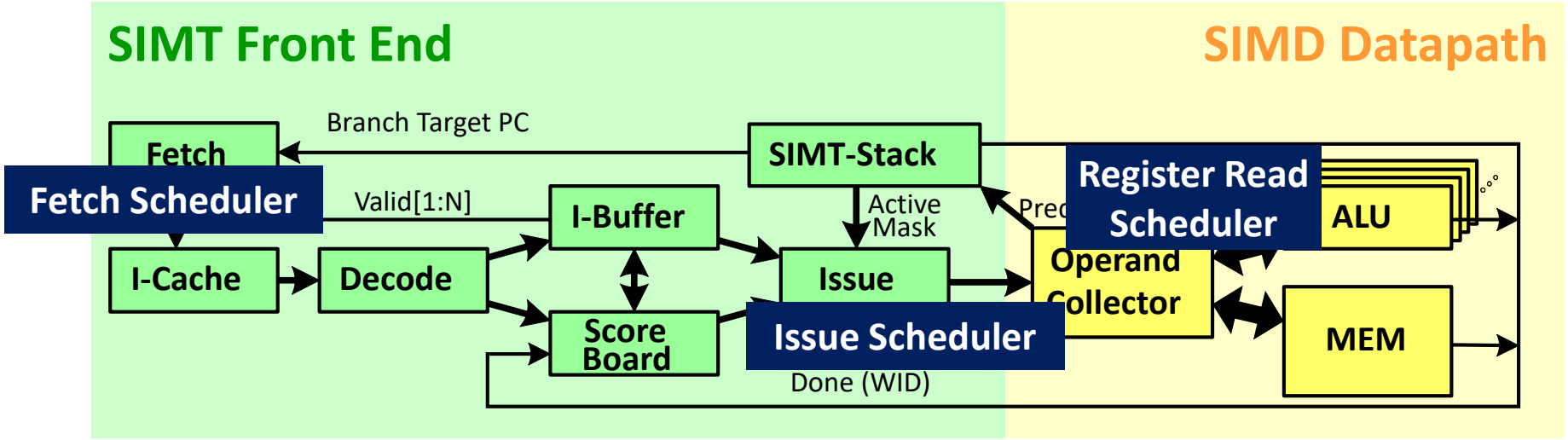
Some of these slides were developed by Tim Rogers at the Purdue University and Tor Aamodt at the University of British Columbia

Slides enhanced by Matt Sinclair

Objectives

- A more detailed scheduler pipeline
  - “Three loop approximation”
  - Register file bank usage scheduler

Final stage of “how to build a GPU pipeline”



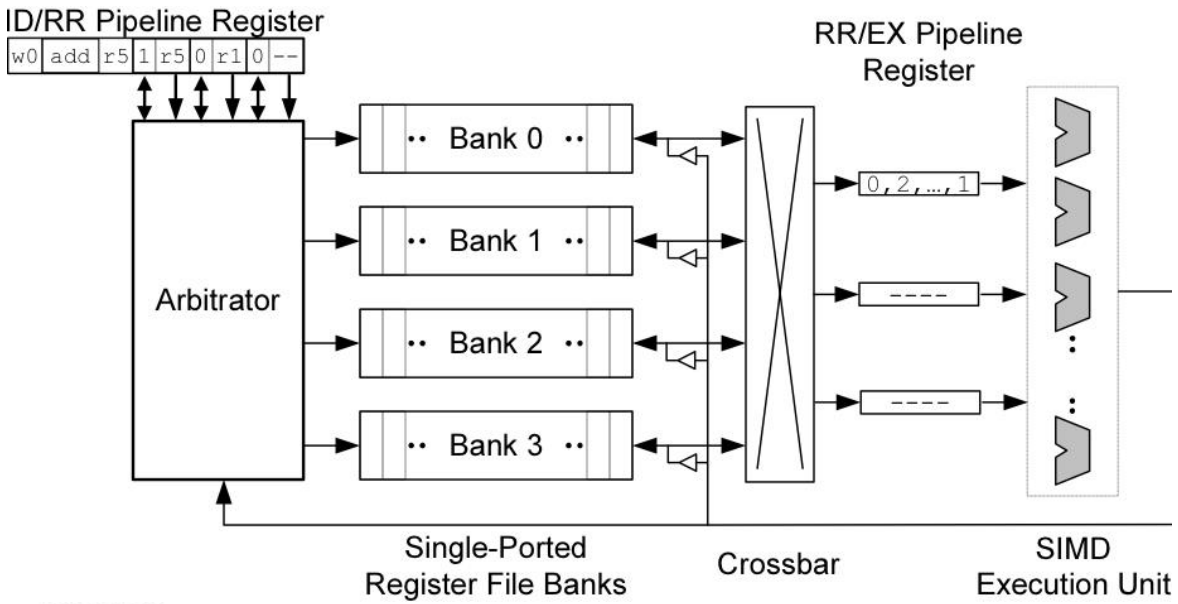
- Final scheduler used to arbitrate register file banks

Register File problem: Size

- All context needs to kept resident
  - Some recent research on context switching, but still very expensive
- Modern GPUs: 256KB of register file
- Need to read multiple values from the register file every cycle to maintain pipeline throughput (FMA instruction: 4 operands – need 4 ports).
- Dual issue to multiple pipelines: need even more ports.
- Big, multi-ported structures are expense in area and energy.

Banked Register File

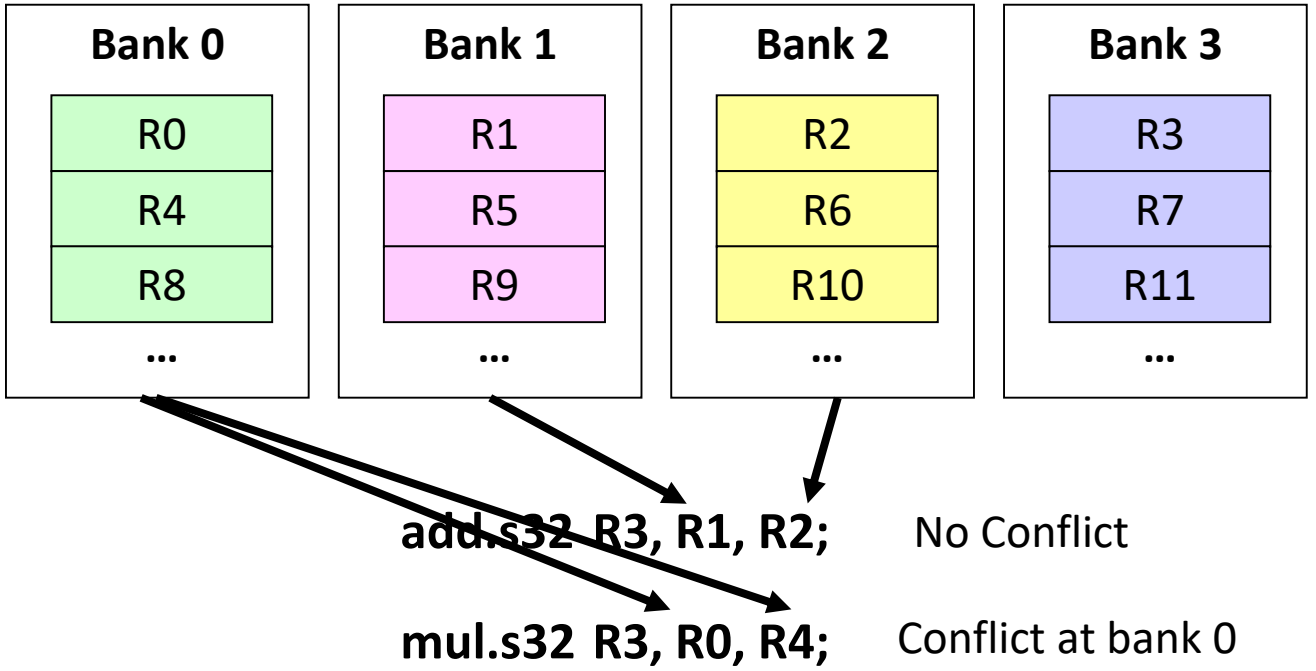
Strawman microarchitecture:



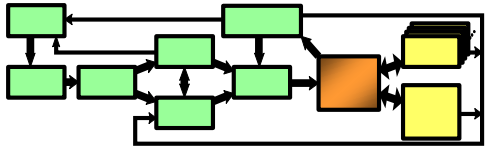
Register layout:

Bank 0	Bank 1	Bank 2	Bank 3
…	…	…	…
w1:r4	w1:r5	w1:r6	w1:r7
w1:r0	w1:r1	w1:r2	w1:r3
w0:r4	w0:r5	w0:r6	w0:r7
w0:r0	w0:r1	w0:r2	w0:r3

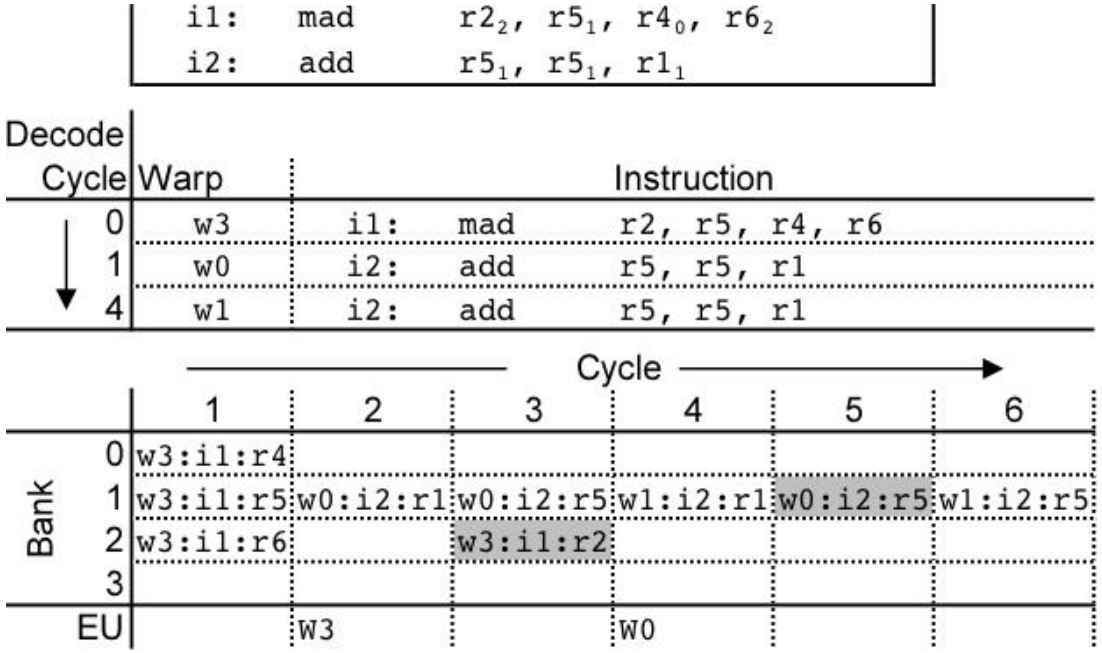
Operand Collector



- Term “Operand Collector” appears in figure in NVIDIA Fermi Whitepaper
- Operand Collector Architecture (US Patent: 7834881)
  - Interleave operand fetch from different threads to achieve full utilization

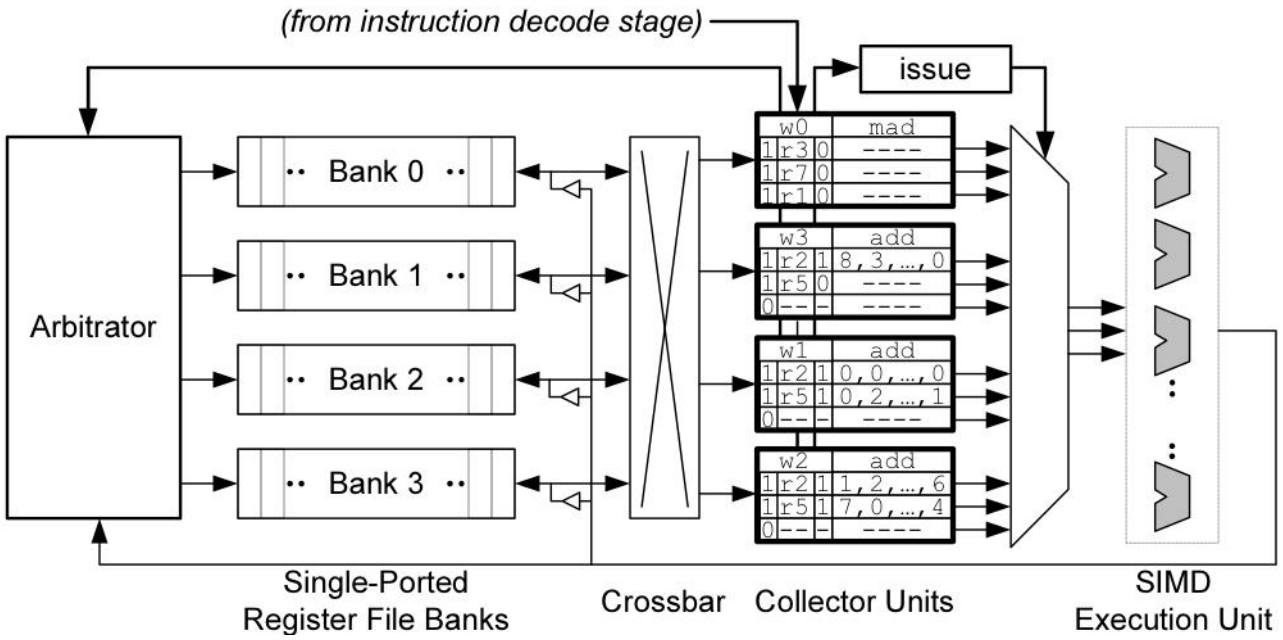


Register Bank Conflicts



- warp 0, instruction 2 has two source operands in bank 1: takes two cycles to read.
- Also, warp 1 instruction 2 is same and is also stalled.
- Can use warp ID as part of register layout to help.

Operand Collector (1)



- Issue instruction to collector unit.
- Collector unit similar to reservation station in tomasulo's algorithm.
- Stores source register identifiers.
- Arbiter selects operand accesses that do not conflict on a given cycle.
- Arbiter needs to also consider writeback (or need read+write port)

Operand Collector (2)

- Combining swizzling and access scheduling can give up to ~ 2x improvement in throughput

```
i1:  add   r1, r2, r5
i2:  mad   r4, r3, r7, r1
```

Cycle	Warp	Instruction
0	w1	i1: add $r1_2$, $r2_3$, $r5_2$
1	w2	i1: add $r1_3$, $r2_0$, $r5_3$
2	w3	i1: add $r1_0$, $r2_1$, $r5_0$
3	w0	i2: mad $r4_0$, $r3_3$, $r7_3$, $r1_1$

Cycle →

Bank	1	2	3	4	5	6
0		w2:r2		w3:r5		w3:r1
1			w3:r2			
2		w1:r5		w1:r1		
3	w1:r2		w2:r5	w0:r3	w2:r1	w0:r7
EU			w1	w2	w3	

Bank 0	Bank 1	Bank 2	Bank 3
…	…	…	…
w1:r7	w1:r4	w1:r5	w1:r6
w1:r3	w1:r0	w1:r1	w1:r2
w0:r4	w0:r5	w0:r6	w0:r7
w0:r0	w0:r1	w0:r2	w0:r3