University of BRISTOL

MSc Computer Science
2022 Summer Project

Modelling Apple's M1 Firestorm Core

Processor Simulation with SimEng

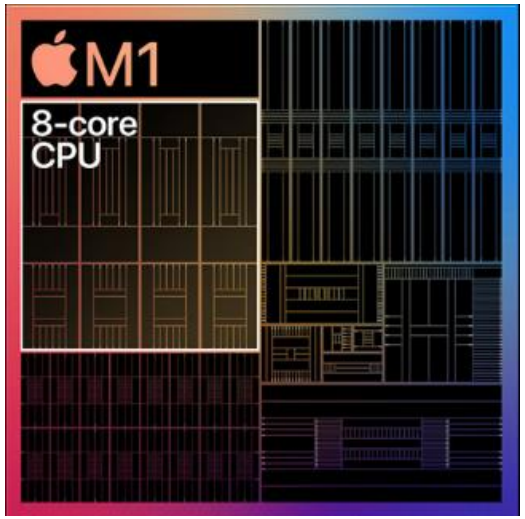
Li-yun Wang

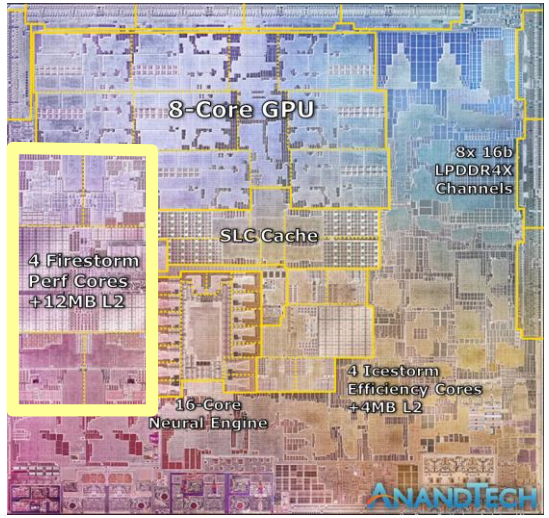
Supervisor: Simon McIntosh-Smith

2022 Sep

Apple's M1 Firestorm

- Apple Inc. introduced its self-designed M1 chip for MacBook in 2020.
  - In M1 CPU, 4 high-performance cores, Firestorm, and 4 high-efficiency cores, Icestorm.
- Firestorm gets impressive performance as compared with its competitor.
  - However, most of the details remain in a black box, disclosed by Apple.





https://www.anandtech.com/show/16226/apple-silicon-m1-a14-deep-dive/2, (2021).

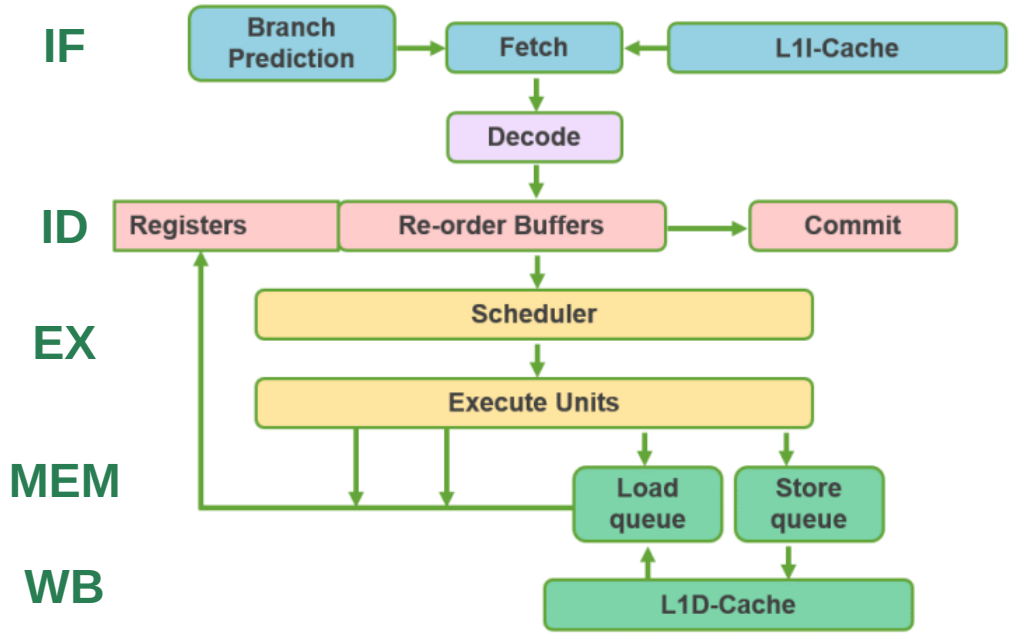
https://www.apple.com/uk/newsroom/2020/11/apple-unleashes-m1/, (2020).

Outline

- Background
- Aims & objectives
- Design & Implementation
  - Configuration
  - Verification
- Analysis & Evaluation
  - Execution Units with Different Opcodes
  - ROB size & register file size
- Future Work
- Conclusion
- Reference

Background

- **Modern Processor microarchitecture**
  - 5-stage pipeline: IF -> ID -> EX -> MEM -> WB
  - Out-of-order
  - Superscalar
- **Processor simulation**
  - Crucial approach for processor design
- **SimEng simulation**
  - A framework developed by Simon's team
  - Supporting sophisticated, superscalar, out-of-order microarchitectures
  - Simple and efficient



https://uob-hpc.github.io/SimEng/, (2021).

D. A. Patterson and J. L. Hennessy, *Computer organization and design ARM edition: the hardware software interface*.

Aim & Objectives

- **Why we need a SimEng simulator for M1 Firestrom?**
  - **M1 Firestorm: renowned, but only a few open source for relevant research**
  - **SimEng simulation: simple, efficient and open source.**

- **In this project,**
  1. **Modelling Apple's M1 Firstorm core with SimEng simulation**
  2. **Verifying the SimEng simulator**
     - **Demonstrate the features of Firestorm**
     - **Accuracy as the simulated cycles being within 10% of Firestorm**
     - **Benchmark with other's research**

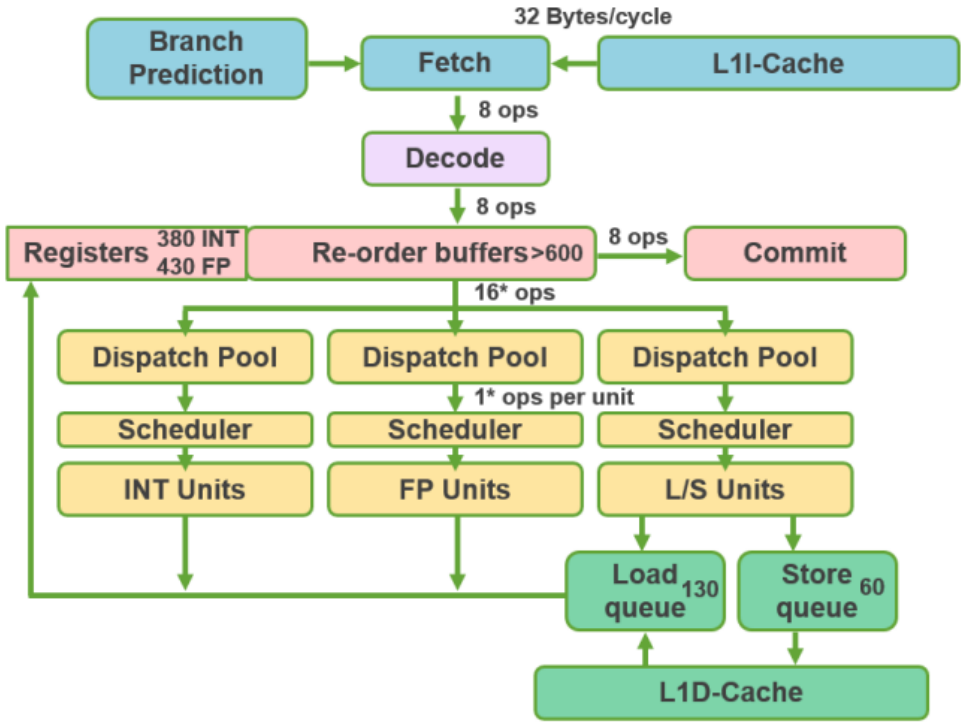
Design and Implementation

1. Configuration of SimEng simulation

- Core
- Bandwidth
- **Buffer size:**
  - Re-order buffer, register files & L/S queue
- **Execution units:**
  - latency, throughput & number of units

2. Verification

- Experiments & program with Assembly codes
  - Loops 1 million times to eliminate noise
- Perf tool: output the clock cycle of real machine



D. Johnson, https://dougallj.github.io/applecpu/firestorm.html, (2021).

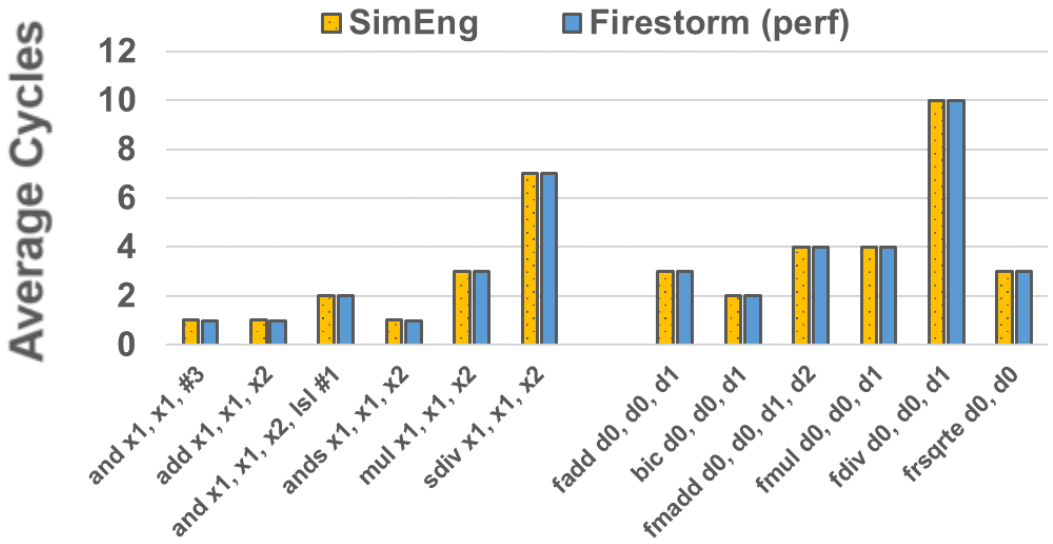
Cardyak, https://drive.google.com/drive/folders/1-Ls0SiqXBCBhMGr87ldA 47GETWfjCzQ, (2022).

Execution Units - Latency

Latency

- **Processor's scalar performance**
  - **Interval to the calculated data ready to other operations**
  - **Shorter, higher performance**
- **Different latency of opcodes**

- **With configuration, SimEng can demonstrate these features.**
  - **Accuracy: miss-matching <1%, goal <10%**



1 cycle for add,
3 cycle for mul
7 cycles for sdiv,

10 cycles for fdiv
4 cycles for ldr

K. Hwang and N. Jotwani, *Advanced computer architecture: parallelism, scalability, programmability*, 1993.

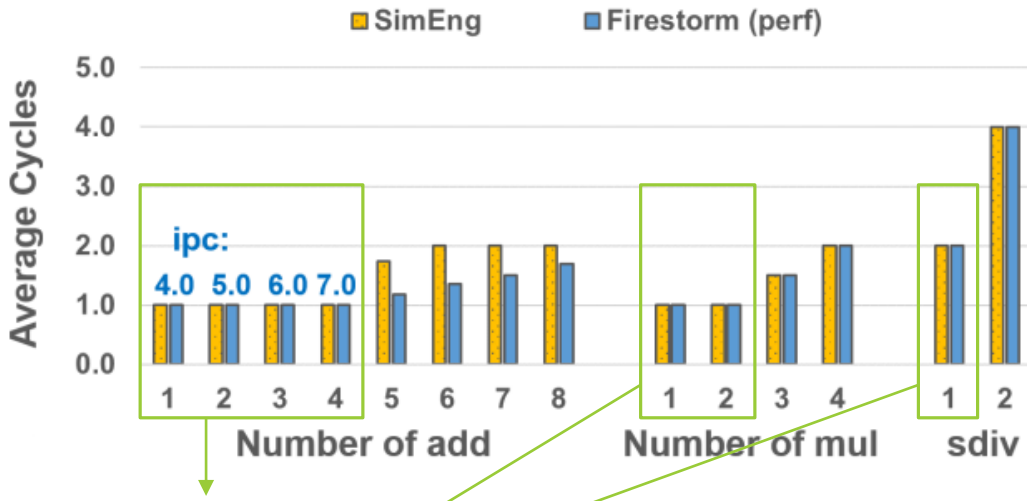
Execution Units – Counts & Throughput

Unit Counts

- Max. number of units can perform the operations

Throughput

- Processor's parallel performance
  - Total instructions completed in a certain time
  - Depending on the number of units
  - higher, higher performance
- Most cases SimEng performs perfectly; however, some finds larger miss-matching ratio >20%



Units
6 units for add*
2 units for mul
1 unit for sdiv

Throughput
1 cycles/insn for add
1 cycles/insn for mul
2 cycles/insn for sdiv

* For add, 4 units with additional 2 units used by loop counter

J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*, 2011.

Execution Units – Accuracy Summary

Accuracy of simulation

- Miss-matching ratio <3%, goal <10%
- Some indep. cases >30% → loop

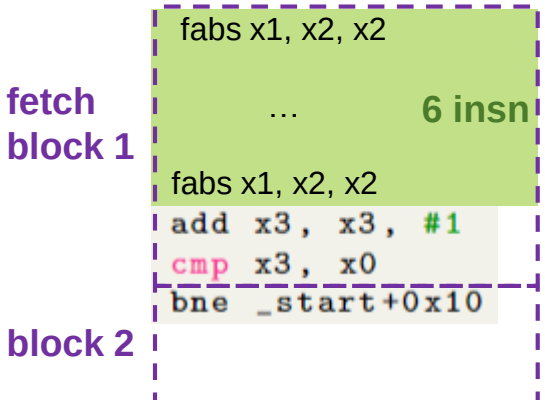
Loop prediction

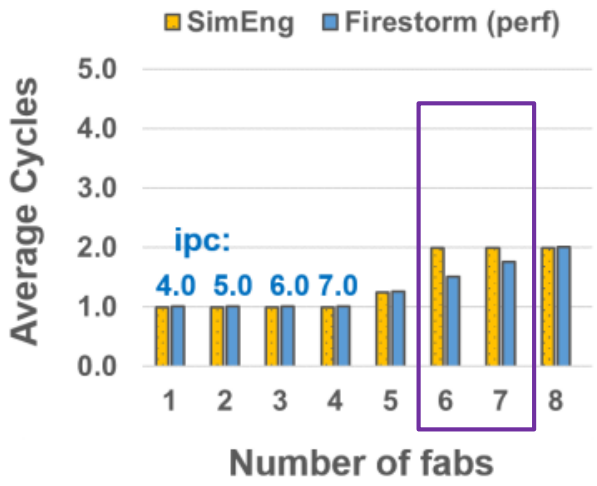
- Firestorm w/ loop prediction
  - After hitting branch, fetch the loop body.
- SimEng ver. w/o loop prediction
  - As the loop body contains > 8 insns, SimEng takes at least 2 cycles.
  - Fetch rate dominates, instead of execution rate.

M. Kobayashi, *Dynamic characteristics of loops*, IEEE Transactions on Computers, 33 (1984), pp. 125–132.

Instruction Group List				
Instruction Groups	opcode	(A)dep.	(B)indep.	(C)indep.
INT_SIMPLE_ARTH_NOSHIFT	add	0%	<1%	0-48%
INT_SIMPLE_LOGICAL_NOSHIFT	and	0%	<1%	0-48%
INT_MUL	mul	0%	<1%	<1%
INT_DIV_OR_SQRT	sdiv	0%	<1%	<1%
FP_SIMPLE_ARTH_NOSHIFT	fabs	0%	<1%	0-33%
FP_SIMPLE_LOGICAL_NOSHIFT	bic	0%	<1%	0-33%
FP_MUL	fmul	0%	<1%	0-33%
FP_DIV_OR_SQRT	fdiv	0%	<1%	<1%
LOAD	ldr	<2%	<1%	0-80%
STORE	str	-	<1%	0-20%

Loop body:





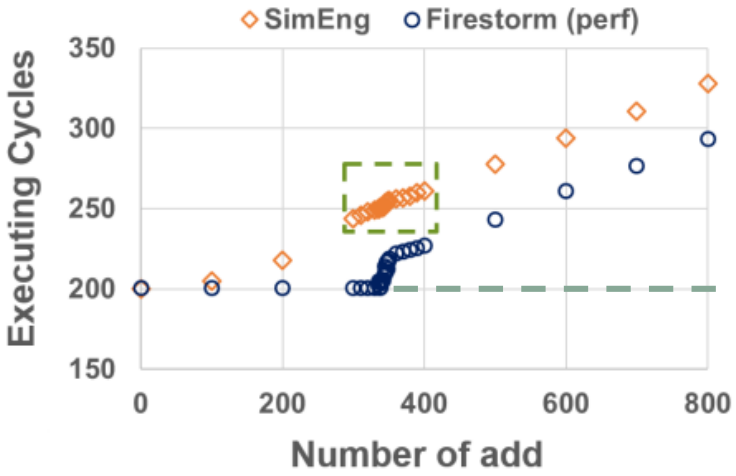
ROB Size & Register File Size

- **Buffers/queues in microarchitecture**
  - Capable of performing superscalar execution
    - Larger, higher potential performance
- **Sizes of buffers**
  - Cycles jump as the buffer is filled

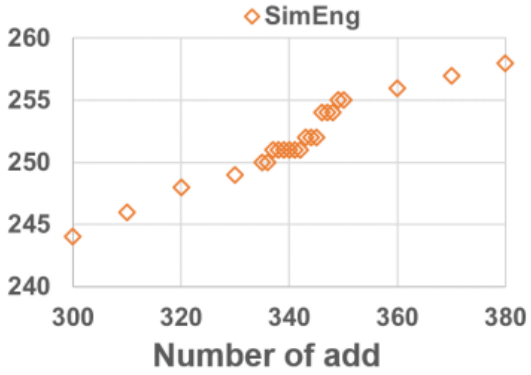
INT register file	FP register file	ROB
344	400	600

  - SimEng can match the position of glitch
  - However, SimEng gets almost linear curve impacted by fetch rate for loops

For INT register file size,

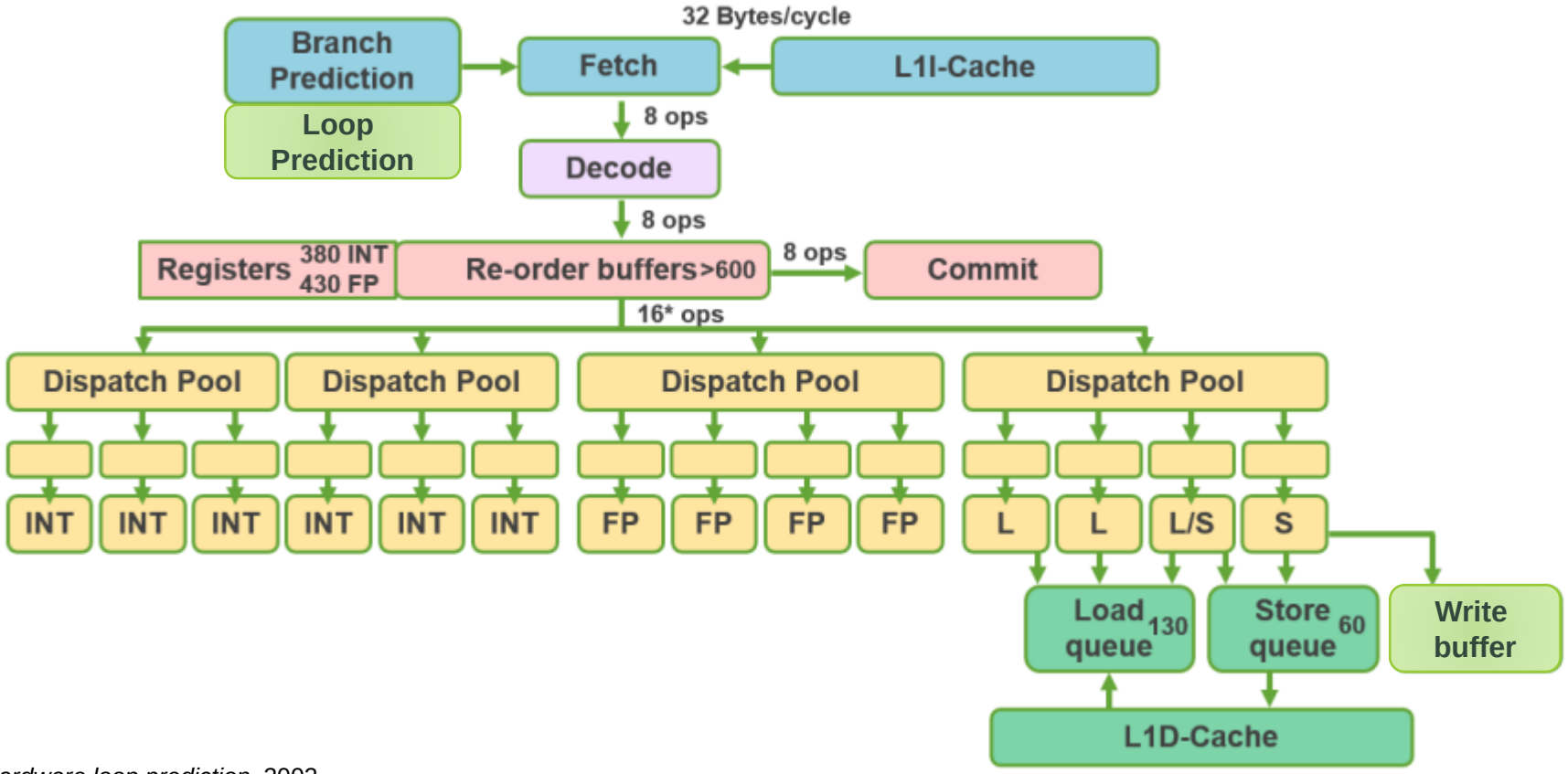


Delay block
fdiv 20 * 10
cycles (latency)



Future Work

- Loop prediction
- Execution units
- Write buffer



M. De Alba and D. Kaeli, *Path-based hardware loop prediction*, 2002.

J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*, 2011.

Conclusion

- **Demonstrate the key features of Firestorm**
  - **Including execution units, latency, throughput, ROB size & register file sizes**
- **Accuracy (goal: <10%)**
  - **For general cases, SimEng can achieve high accuracy (miss ratio <3%)**
  - **Worse cases impacted by loop prediction**
  - **Overall miss-matching ratio 25% of Firestorm**

Thank you

Reference

- *Apple unleashes m1*, https://www.apple.com/uk/newsroom/2020/11/apple-unleashes-m1/, (2020).
- A. Frumusanu, *Apple announces the apple silicon m1: Ditching x86 - what to expect, based on a14*, https://www.anandtech.com/show/16226/apple-silicon-m1-a14-deep-dive/2, (2021).
- *The simulation engine - simeng documentation*, https://uob-hpc.github.io/SimEng/, (2021).
- D. A. Patterson and J. L. Hennessy, *Computer organization and design ARM edition: the hardware software interface*, Morgan kaufmann, 2016.
- J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*, Elsevier, 2011.
- K. Hwang and N. Jotwani, *Advanced computer architecture: parallelism, scalability, programmability*, vol. 199, McGraw-Hill New York, 1993.
- Cardyak, *Cpu march block diagrams*, https://drive.google.com/drive/folders/1-Ls0SiqXBCBhMGr87ldA47GETWfjCzQ, (2022).
- M. Kobayashi, *Dynamic characteristics of loops*, IEEE Transactions on Computers, 33 (1984), pp. 125–132.
- M. De Alba and D. Kaeli, *Path-based hardware loop prediction*, in 4th International Conference on Control, Virtual Instrumentation and Digital Systems, 2002, pp. 29–38.

Backup

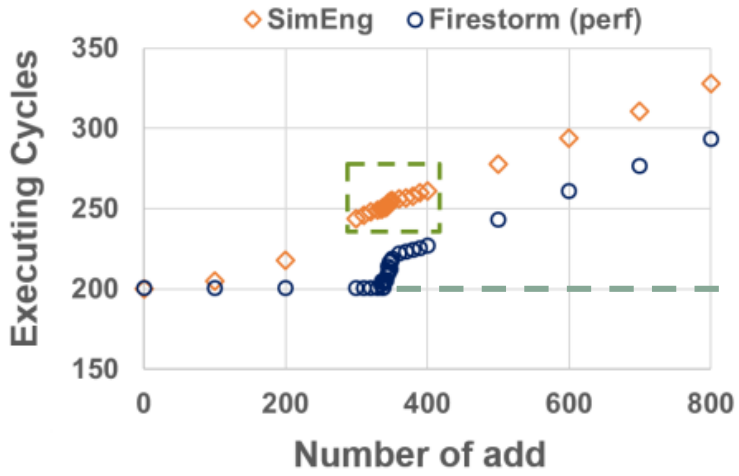
ROB Size & Register File Size

- **Buffers/queues in microarchitecture**
  - Capable of performing superscalar execution
    - Larger, higher potential performance
- **Sizes of buffers**
  - Cycles jump as the buffer is filled

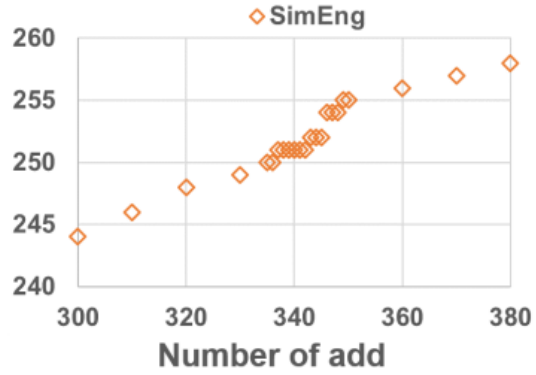
	INT register file	FP register file	ROB
My work	344	400	600
Dougall J.	380	432	623
Potential miss-match	**9.7%**	**8.0%**	**3.7%**

- SimEng can match the position of glitch
- However, SimEng gets almost linear curve impacted by fetch rate for loops

For INT register file size,



Delay block
fdiv 20 * 10
cycles (latency)



D. Johnson, https://dougallj.github.io/applecpu/firestorm.html, (2021).

Program: Hailstone Sequence

Verifying SimEng with an actual program

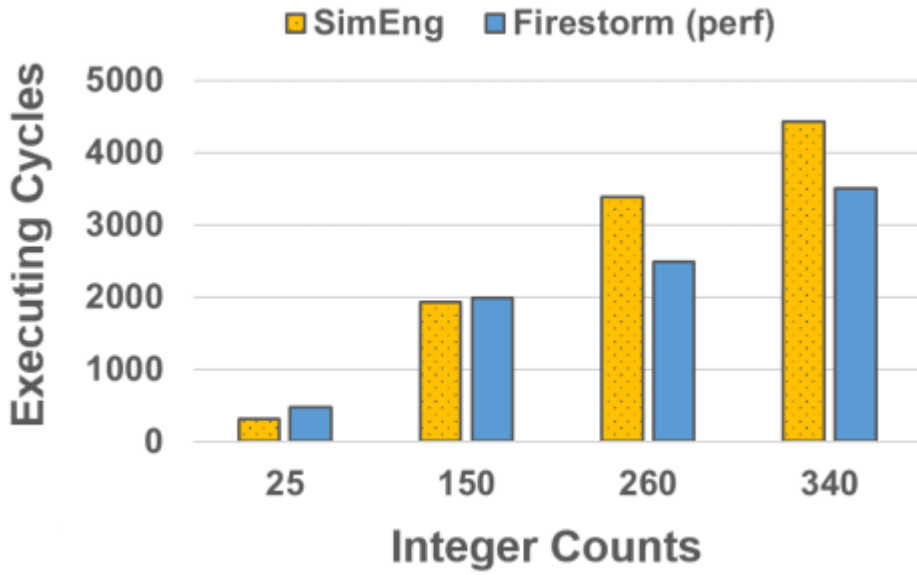
- Generating integers w/ rules
  - Every integer takes at least 2 branches

```
if( x % 2 == 0 ) {
    x = x / 2 ;
}
else
    x = 3 * x + 1 ;
```

branch

branch

- SimEng match as generating 150 integers
- SimEng's accuracy ↓ as integer counts ↓
  - Overall accuracy: 25%, goal <10%
  - Possibly impacted by loop prediction or branch miss-prediction



Assembly

```
1  .global _start
2  .align 2
3  _start:
4              mov x0, 1
5              mov x1, 2
6              mov x0, #1000
7              mul x0, x0, x0          @ iteration times
8              add x1, x1, #3          @
9              add x1, x1, #3          @ main body
10             add x1, x1, #3          @
11             add x3, x3, #1          @ loop counter
12             cmp x3, x0
13             bne _start+0x10         @ conditional branch
14             mov x0, #0              @ system call
15             mov x8, #94
16             svc #0
```

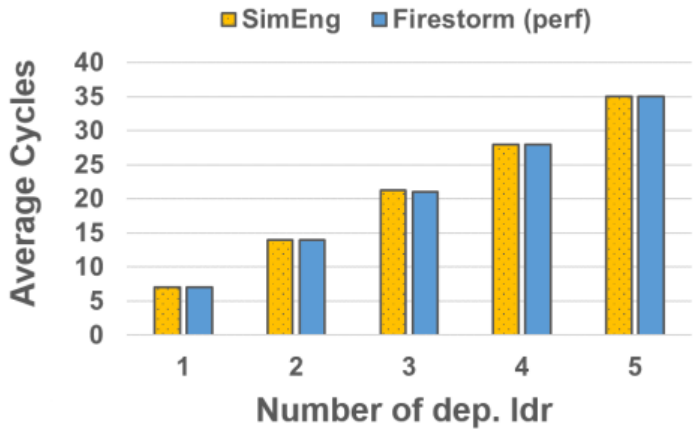
Latency, throughput, and unit counts of opcode groups

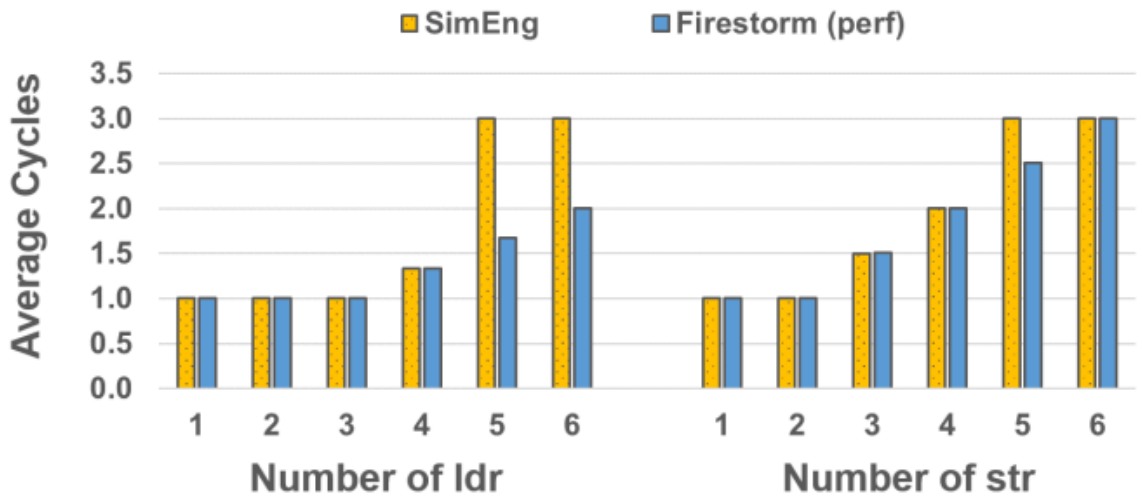
Instruction Group List				
Instruction Groups	Opcodes	Latency	Through-put	Unit Counts
INT_SIMPLE_ARTH_NOSHIFT	add	1	1	6
INT_SIMPLE_LOGICAL_NOSHIFT	and	1	1	6
INT_SIMPLE_CMP	cmp	1	1	3
INT_MUL	mul	3	1	2
INT_DIV_OR_SQRT	sdiv	7	2	1
FP_SIMPLE_ARTH_NOSHIFT	fabs	2	1	4
FP_SIMPLE_LOGICAL_NOSHIFT	bic	2	1	4
FP_MUL	fmul	4	1	4
FP_DIV_OR_SQRT	fdiv	10	1	1

Loop Prediction

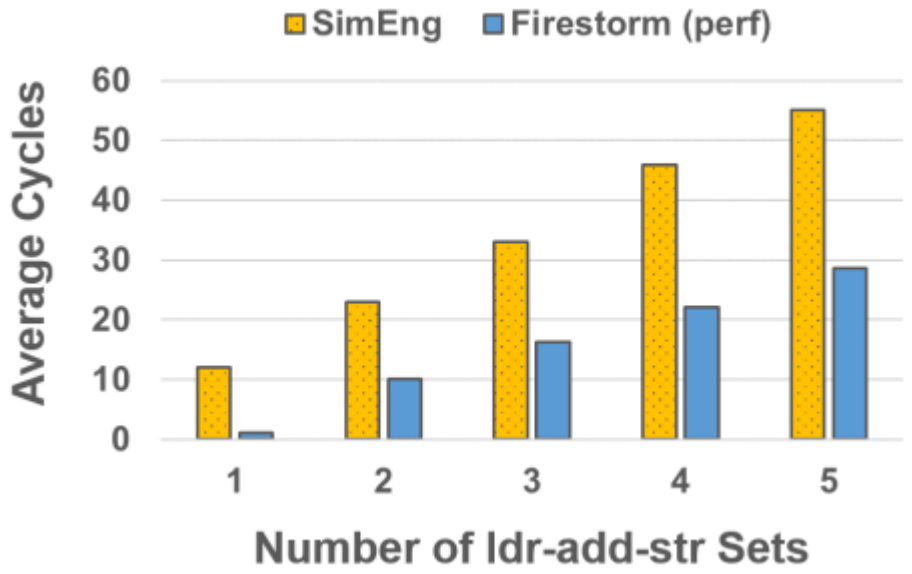
```
1  .global _start
2  .align 2
3  _start:
4              mov x0, 1
5              mov x1, 2
6              mov x0, #1000
7              mul x0, x0, x0          @ iteration times
8              add x1, x1, #3          @
9              add x1, x1, #3          @ main body
10             add x1, x1, #3          @
11             add x3, x3, #1          @ loop counter
12             cmp x3, x0
13             bne _start+0x10         @ conditional branch
14             mov x0, #0              @ system call
15             mov x8, #94
16             svc #0
```

L/S – Latency & Throughput

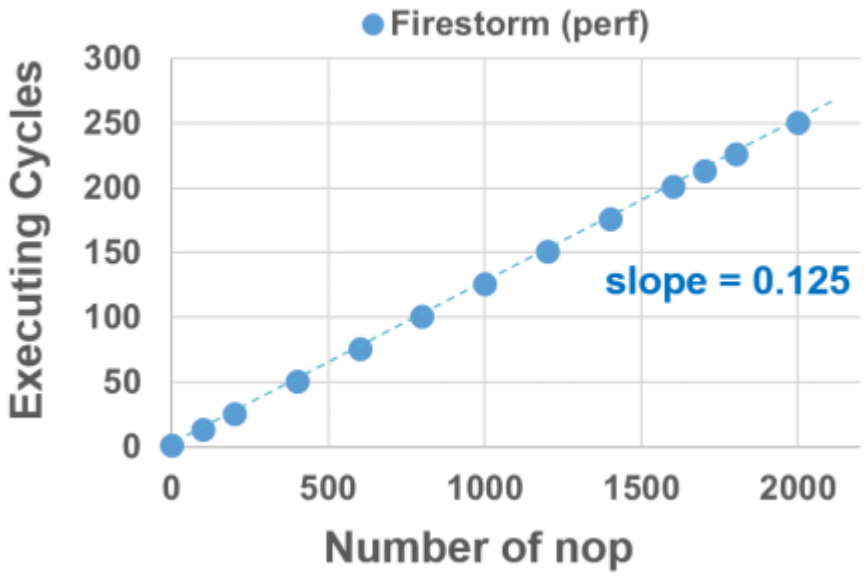




Ldr-str Depedency

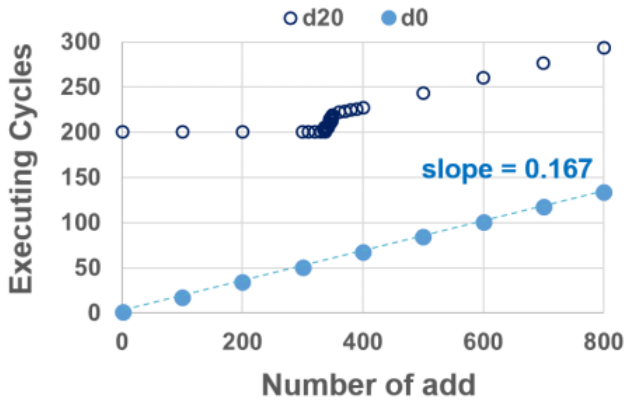


Bandwidth

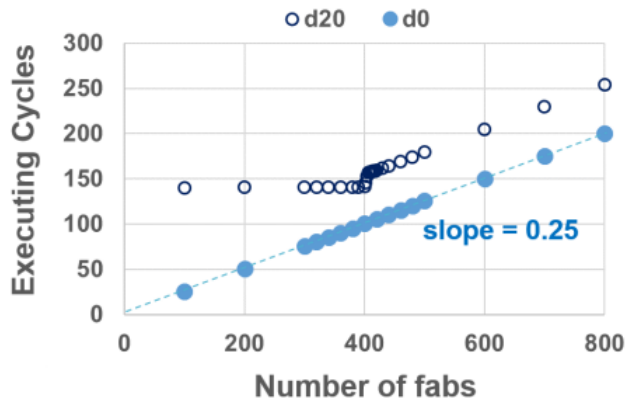

Firestorm (perf)
Executing Cycles
300
250
200
150
100
50
0
0
500
1000
1500
2000
Number of nop
slope = 0.125

Perf results of Buffer Size

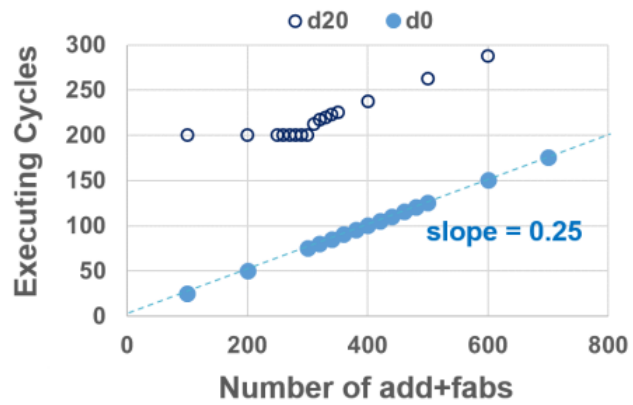
Integer register file



FP register file



ROB



Hailstone Sequence

```
.global _start
.align 2
_start:
  mov x9, #1
  mul x9, x9, x9
  mov x10, #0
_loop:
  mov x7, #3591
        cmp x7, #1
        beq _end
        tst x7, #1
        beq _even
        bne _odd
_even:
        mov x6, #2
        udiv x4, x7, x6
        mov x7, x4
        b _loop+0x4
_odd:
        mov x5, #3
        mul x3, x7, x5
        add x7, x3, #1
        b _loop+0x4
_end:
  add x10, x10, #1
  cmp x10, x9
  bne _loop
        mov x0, #0
        mov x8, #94
        svc #0
```