Nvidia Denver architecture

And some NEON

The contents of this presentation

- A bit about CPUs
- Scalar code vs SIMD
- Explaining the Denver architecture
- Some tips and tricks

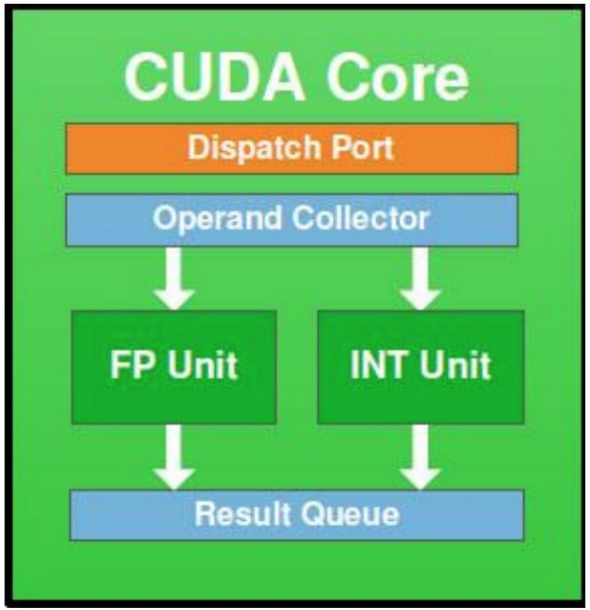
About me

- Dino Orucevic
- Recently finished my Master’s thesis
- May be of interest to you!

CPUs and SIMD

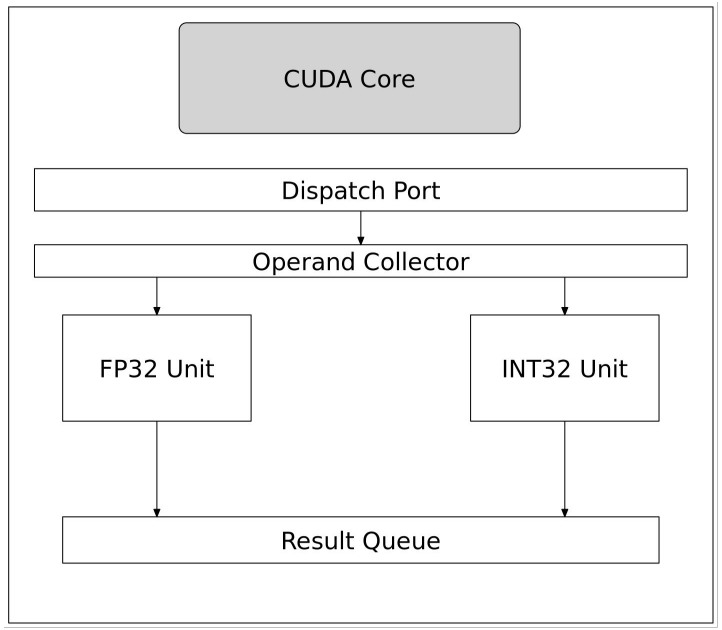
CUDA Core

- You may have already seen that this is a CUDA core



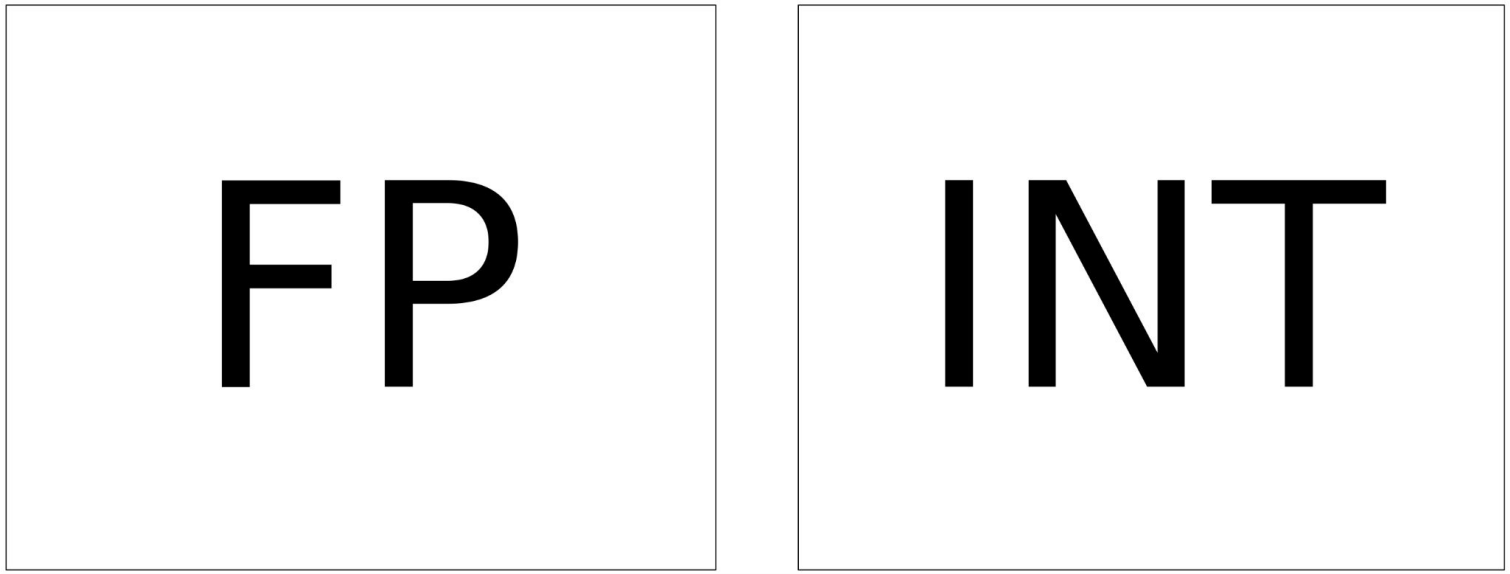
CUDA Core

- You may have already seen that this is a CUDA core



The CPU has ALUs/Pipelines too

- A CPU core also has ALUs
  - FPUs and Integer units



The ALUs have a lot of functions

- Bitshifts
- ADD
- MUL
- MAC
- DIV
- ARM NEON

The ALUs have a lot of functions

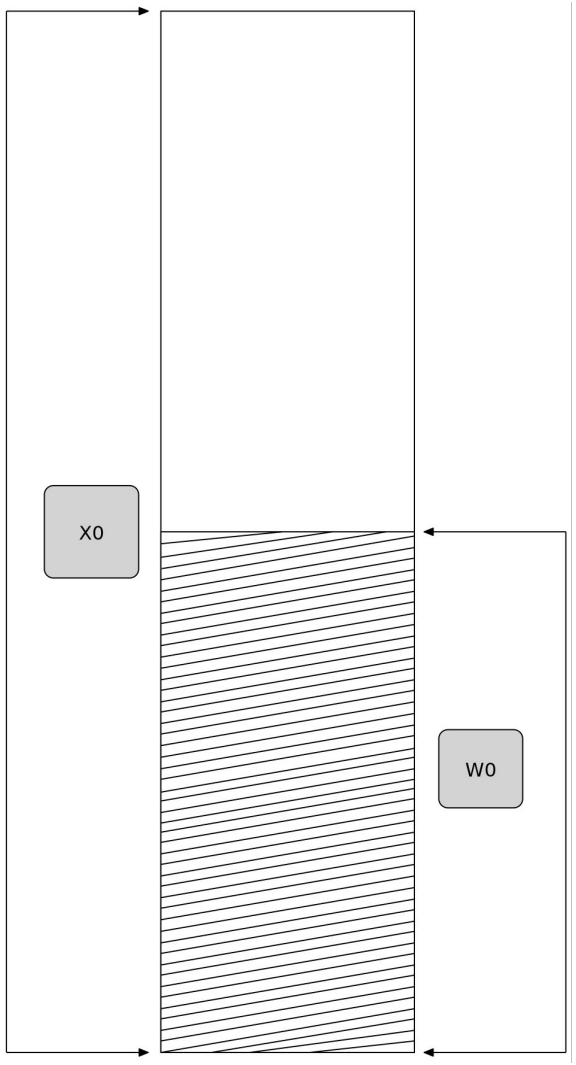
- Bitshifts
- ADD
- MUL
- MAC
- DIV
- ARM NEON
- Not every ALU is identical!
    - Not all functionality is included

And the CPU ALUs are different

- CPU architecture
  - Branching
    - Less parallelism
  - Certain operations
    - Casting
  - Sequential code

And the CPU ALUs are different

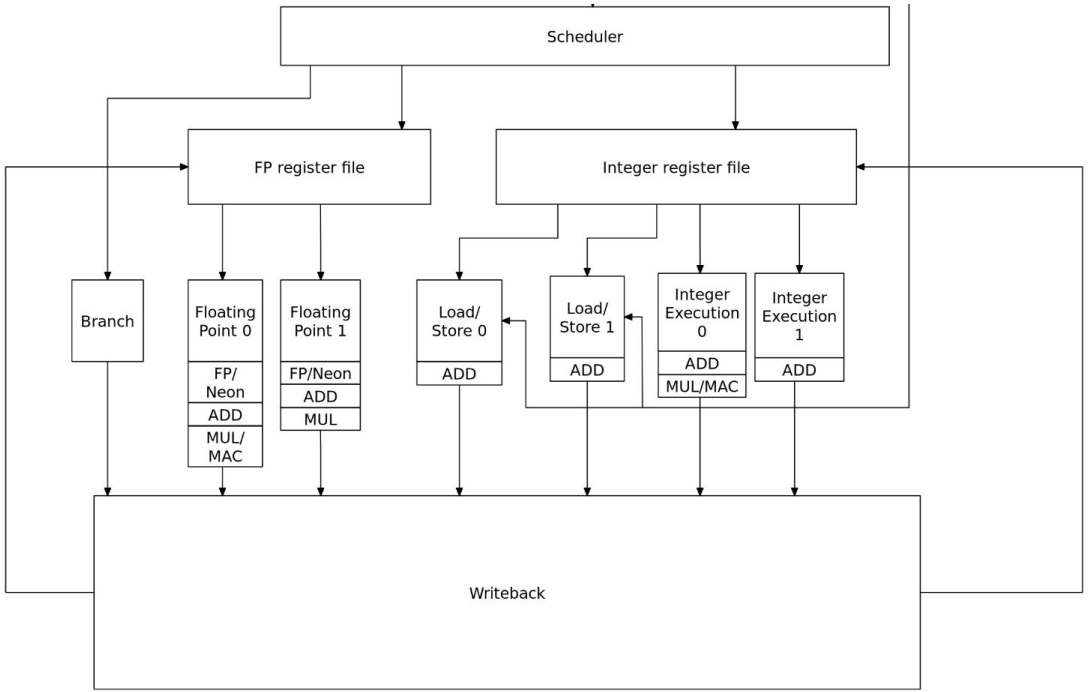
- CPU architecture
  - Branching
    - Less parallelism
  - Certain operations
    - Casting
  - Sequential code
- General purpose registers
  - 64-bit
  - Can use any data type*



And the CPU ALUs are different

- CPU architecture
  - Branching
    - Less parallelism
  - Certain operations
    - Casting
  - Sequential code
- General purpose registers
  - 64-bit
  - Can use any data type*
- Separated ALUs
  - MIMD
  - Can be used in parallel

Cortex A57 back-end

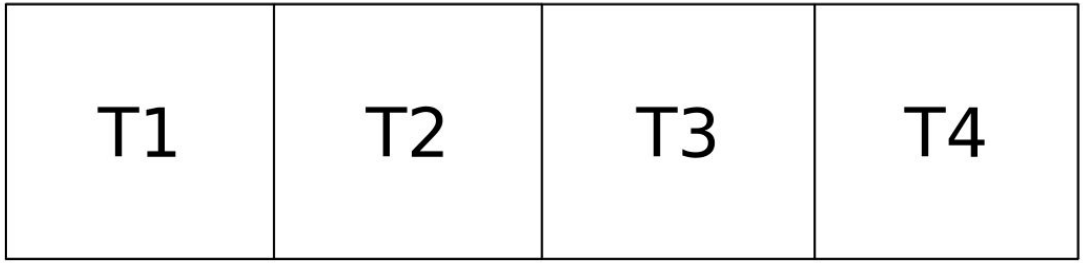


SIMT

a*b = res_one

c*d = res_two

e*f = res_three

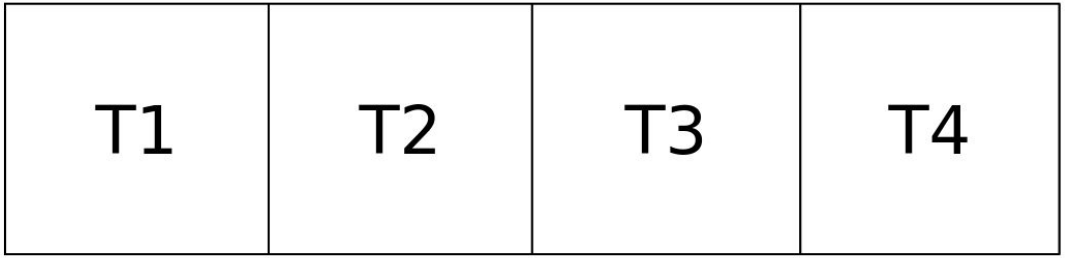


SIMT

a*b = res_one

c*d = res_two

e*f = res_three

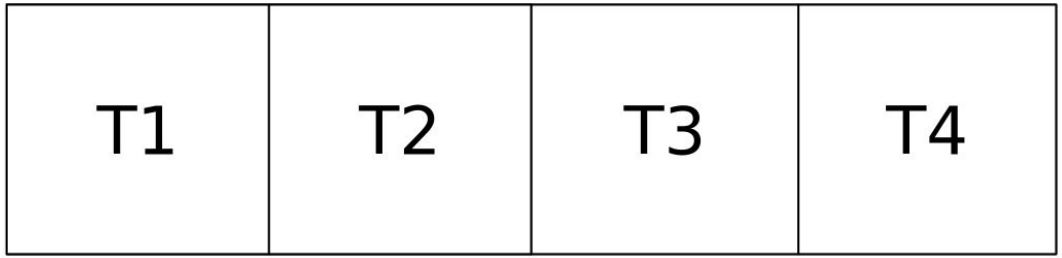


SIMT

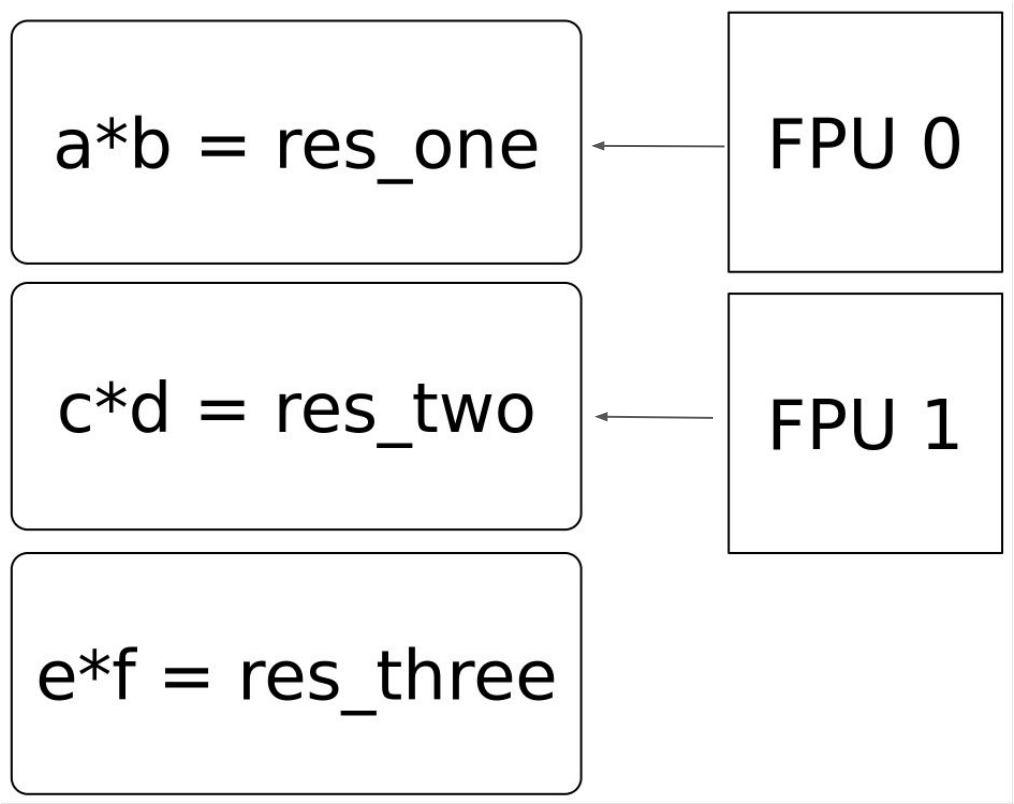
a*b = res_one

c*d = res_two

e*f = res_three



MIMD



This sounds great!

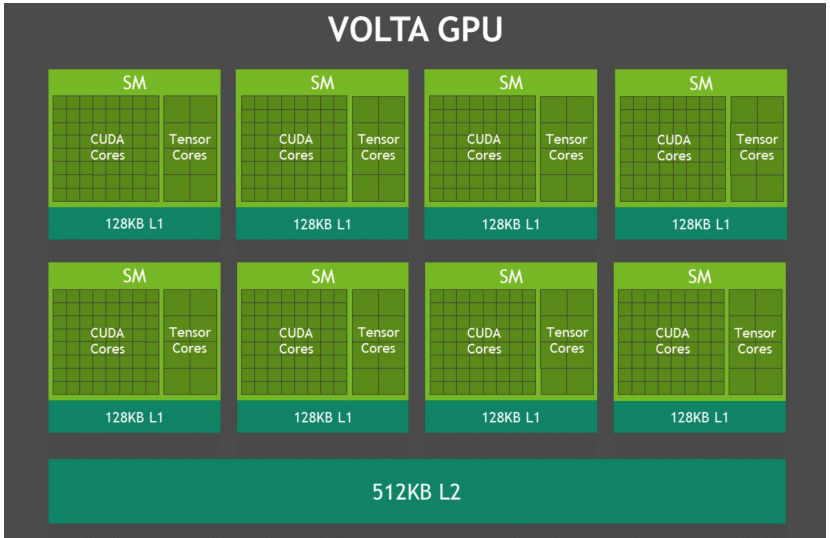
Why do even need a GPU?

Nothing comes for free

- Complex CPU/ALUs = Bigger cores/ALUs
  - Less power efficient
- Less FPUs
  - Only two FPUs per core in Denver/Carmel

Nothing comes for free

- Complex CPU/ALUs = Bigger cores/ALUs
  - Less power efficient
- Less FPUs
  - Only two FPUs per core in Denver/Carmel



Nothing comes for free

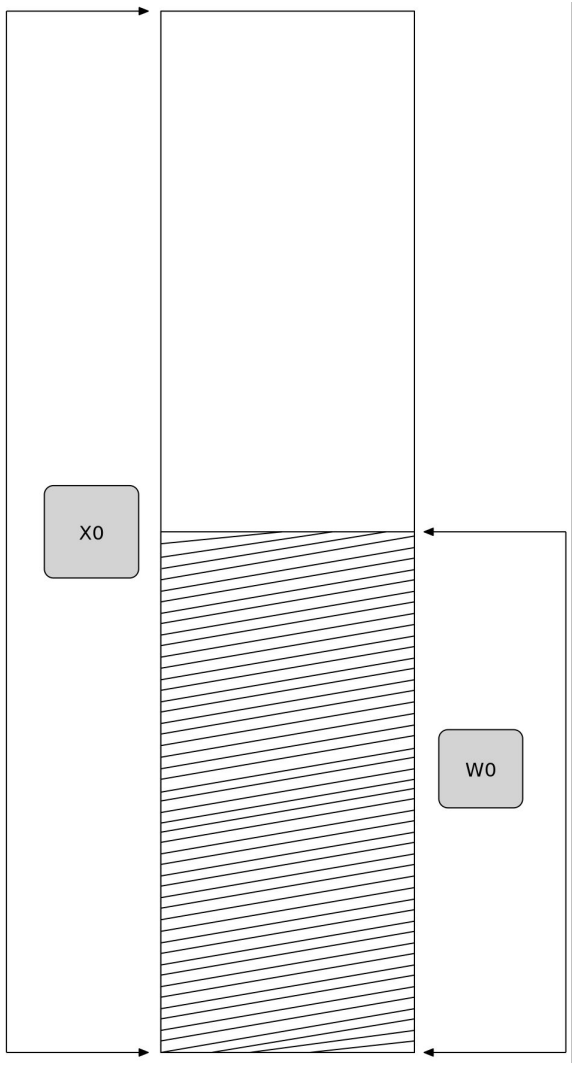
- Complex CPU/ALUs = Bigger cores/ALUs
  - Less power efficient
- Less FPUs
  - Only two FPUs per core in Denver/Carmel
  - Could we utilize the FPUs better?

SIMD

Single Instruction Many Data

SIMD

- Special hardware and registers
- Each register = 128-bit (NEON)
- What changes?
  - Scalar code
  - A register = a single variable



SIMD

- Special hardware and registers
- Each register = 128-bit (NEON)
- Can be broken up into:
  - 2x64 bit
  - 4x 32 bit
  - 8x 16 bit
  - 16x8 bit (int)



SIMD

- Special hardware and registers
- Each register = 128-bit (NEON)
- Can be broken up into:
  - 2x64 bit
  - 4x 32 bit
  - 8x 16 bit
  - 16x8 bit (int)
- 1 instruction
  - Affects all of the variables in the register



SIMD

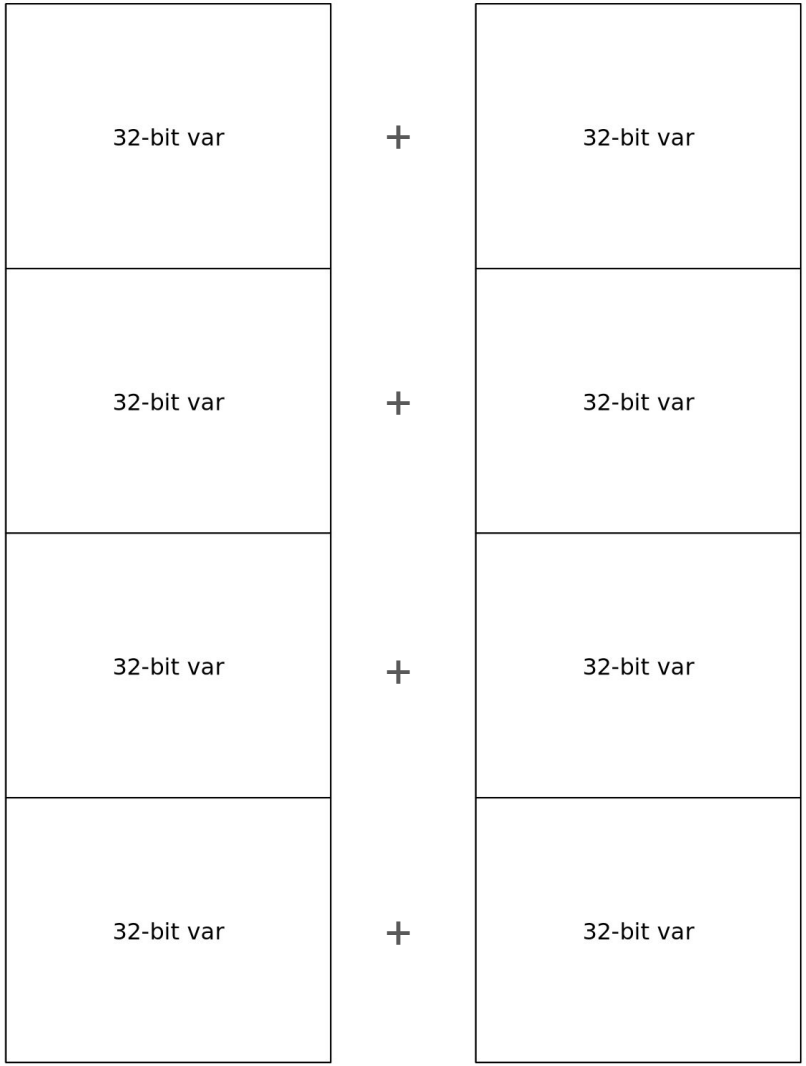
- Special hardware and registers
- Each register = 128-bit (NEON)
- Can be broken up into:
  - 2x64 bit
  - 4x 32 bit
  - 8x 16 bit
  - 16x8 bit (int)
- 1 instruction
  - Affects all of the variables in the register

```
vaddq_f32
```



SIMD

`vaddq_f32`



Denver/ Carmel

What is Denver?

- Nvidia’s CPU microarchitecture
- Had 3 generations
  - Denver (64-bit K1)
    - 2014
  - Denver 2 (TX2)
    - 2017
  - Carmel (Xavier)
    - 2018



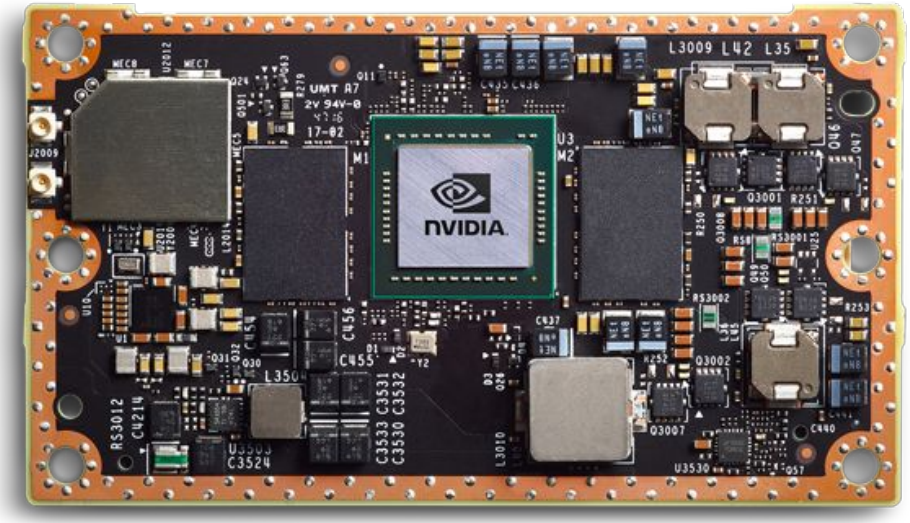
What is Denver?

- Nvidia’s CPU microarchitecture
- Had 3 generations
  - Denver (64-bit K1)
    - 2014
  - Denver 2 (TX2)
    - 2017
  - Carmel (Xavier)
    - 2018



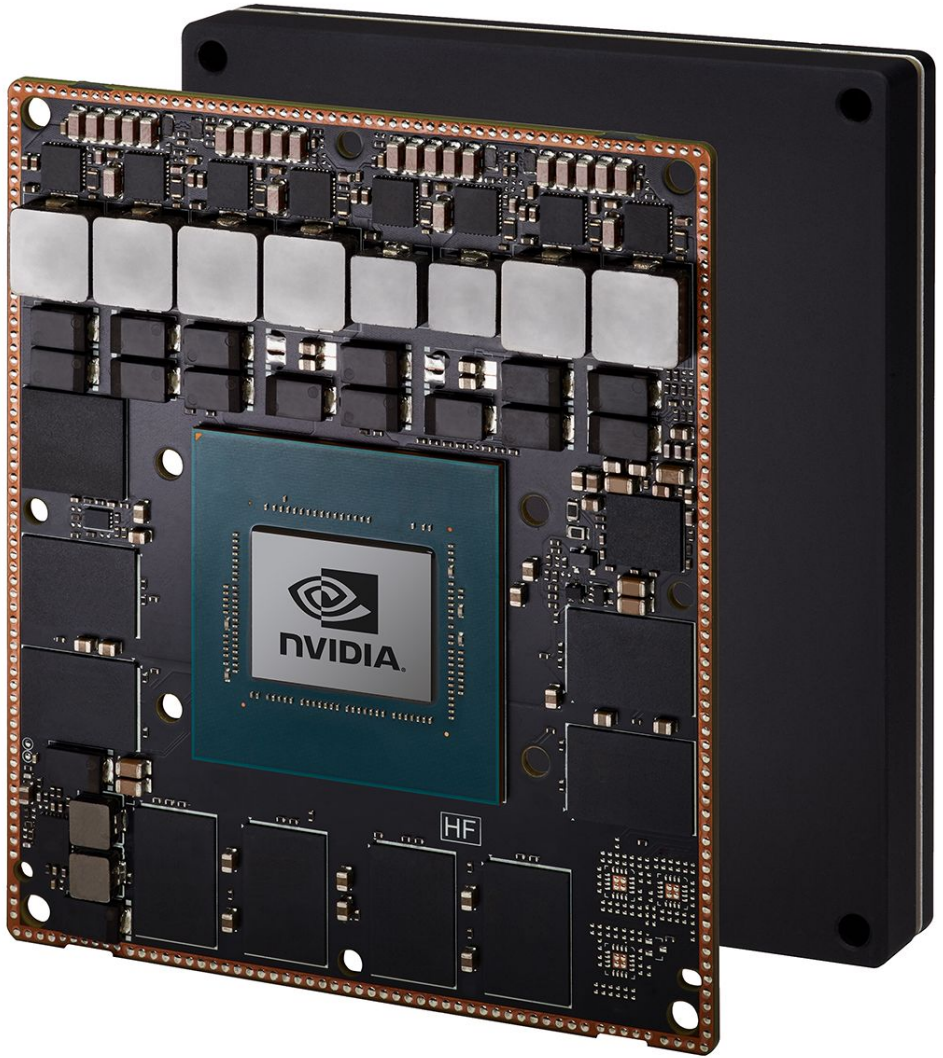
Jetson TX2 - specifications

- CPU
  - 4x Cortex-A57 cores
  - 2x Denver 2 cores
- GPU
  - Pascal μarch
  - 256 CUDA Cores
  - 1300MHz
- Memory
  - 8 GB
- Power budget
  - 15W



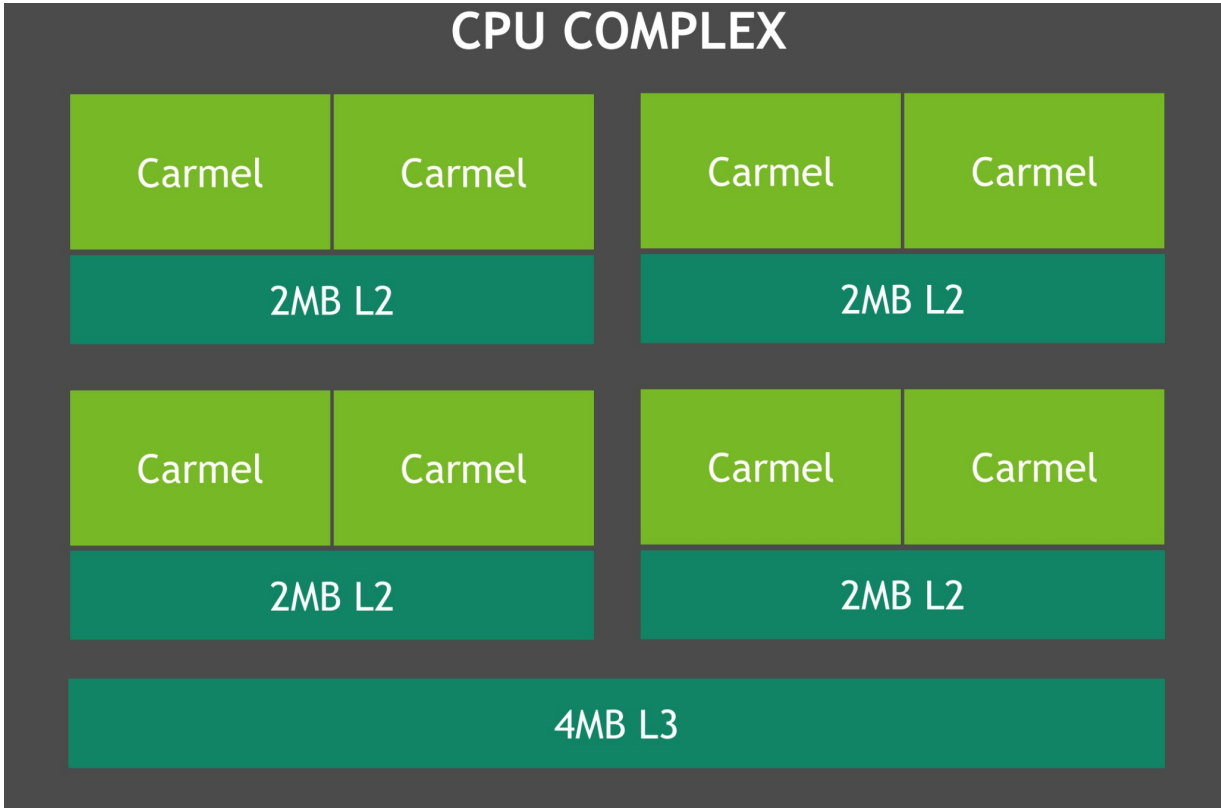
Jetson Xavier - Specifications

- CPU
  - 8x Carmel cores
- GPU
  - Volta μarch
  - 512 CUDA Cores
  - 1357MHz
- Memory
  - 32 GB
- Power budget
  - 30W



Jetson Xavier - Specifications

- CPU
  - 8x Carmel cores
- GPU
  - Volta μarch
  - 512 CUDA Cores
  - 1357MHz
- Memory
  - 32 GB
- Power budget
  - 30W



The Denver architecture

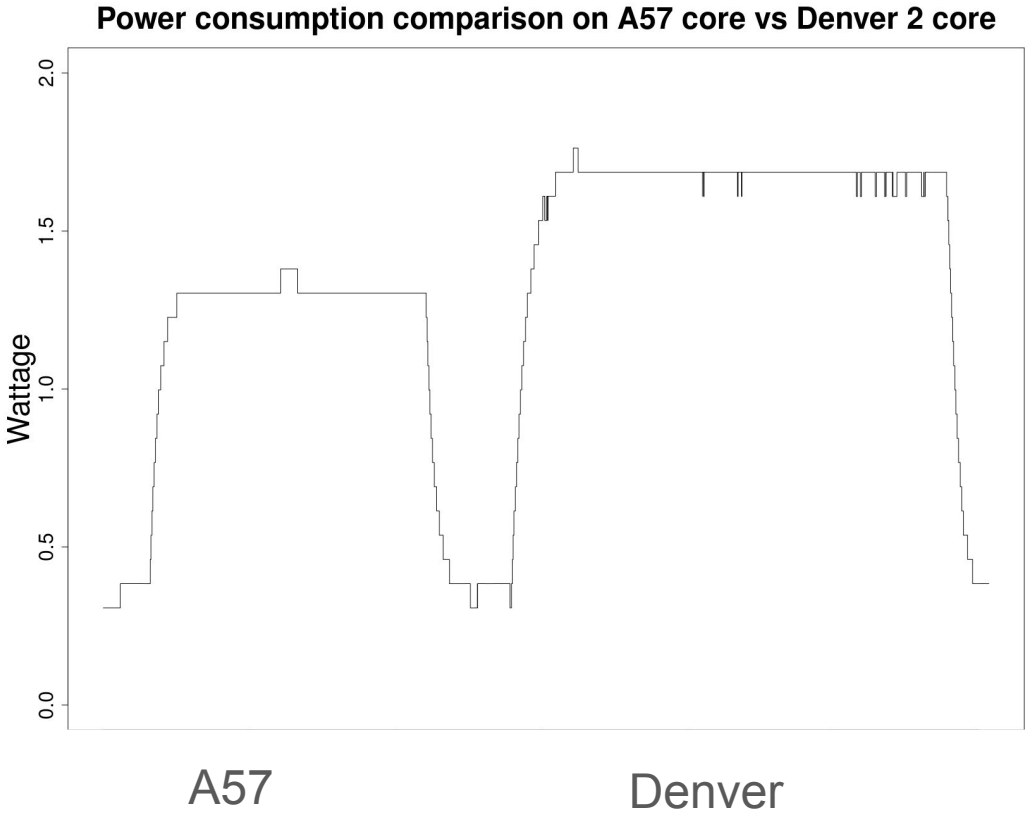
Denver 2 (TX2)

- High performance

Cortex A57

- Low power/ High performance

Power consumption Denver 2 vs A57 single core



- Carmel?

The Denver architecture

Denver 2 (TX2)

- High performance
- Less conventional

Cortex A57

- Low power/ High performance
- A more conventional CPU

The Denver architecture

Denver 2 (TX2)

- High performance
- Less conventional
  - Not out of order

Cortex A57

- Low power/ High performance
- A more conventional CPU
  - Out of order

The Denver architecture

Denver 2 (TX2)

- High performance
- Less conventional
  - Not out of order
  - DCO
    - Similar to out of order

Cortex A57

- Low power/ High performance
- A more conventional CPU
  - Out of order

The Denver architecture

Denver 2 (TX2)

- High performance
- Less conventional
  - Not out of order
  - DCO
    - Similar to out of order
    - Can Optimize code
    - FP
      - A32
    - Int
      - Strength reduction

Cortex A57

- Low power/ High performance
- A more conventional CPU
  - Out of order

Some tests and results

Integer

Calculating test execution time

Number of instructions in parallel

TX2

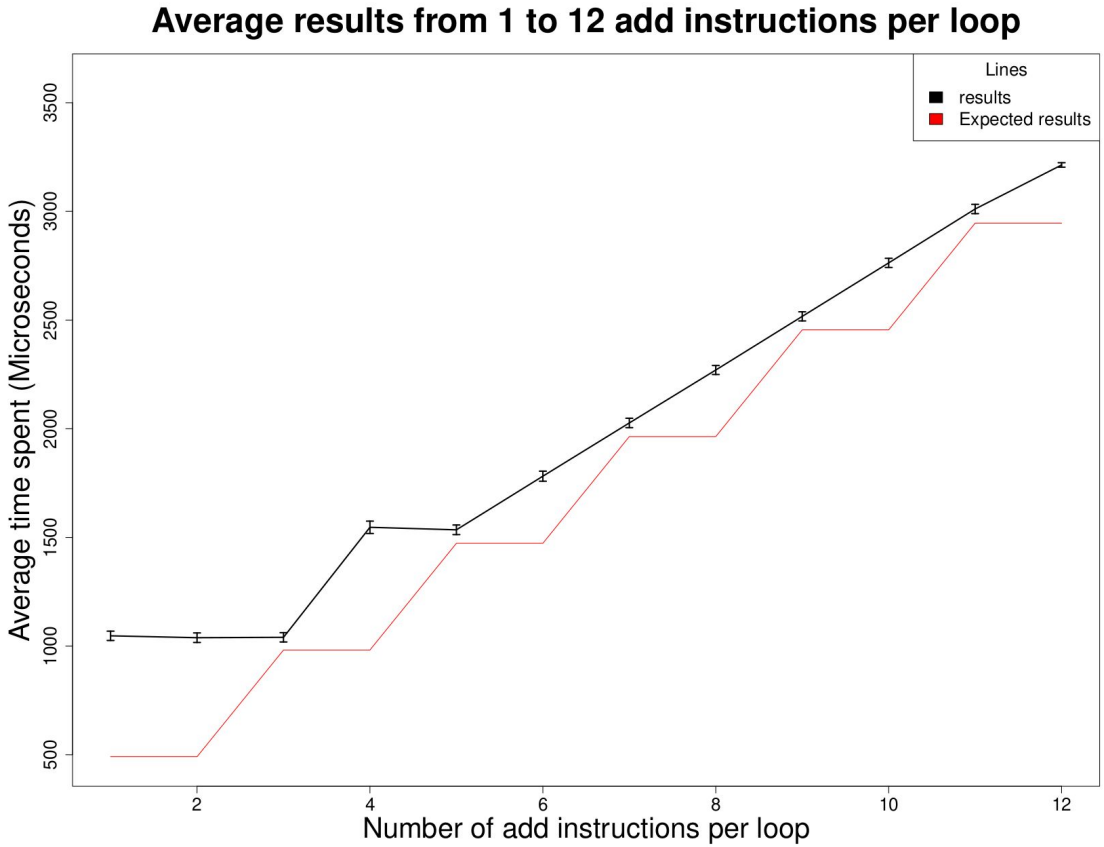
- Number of seconds per cycle:
  - 2035MHz = 2.035*10^9
  - 1/(2.035*10^9) = 4.91*10^(-10)
- Number NS per cycle
  - 4.91*10^(-10)/(10^(-9)) = 0.491ns
- Convert to MS
  - 0.491/1000 = 0.000491
- 1 million cycles
  - 0.000491*1000000 = 491 MicroSec

Number of instructions in parallel

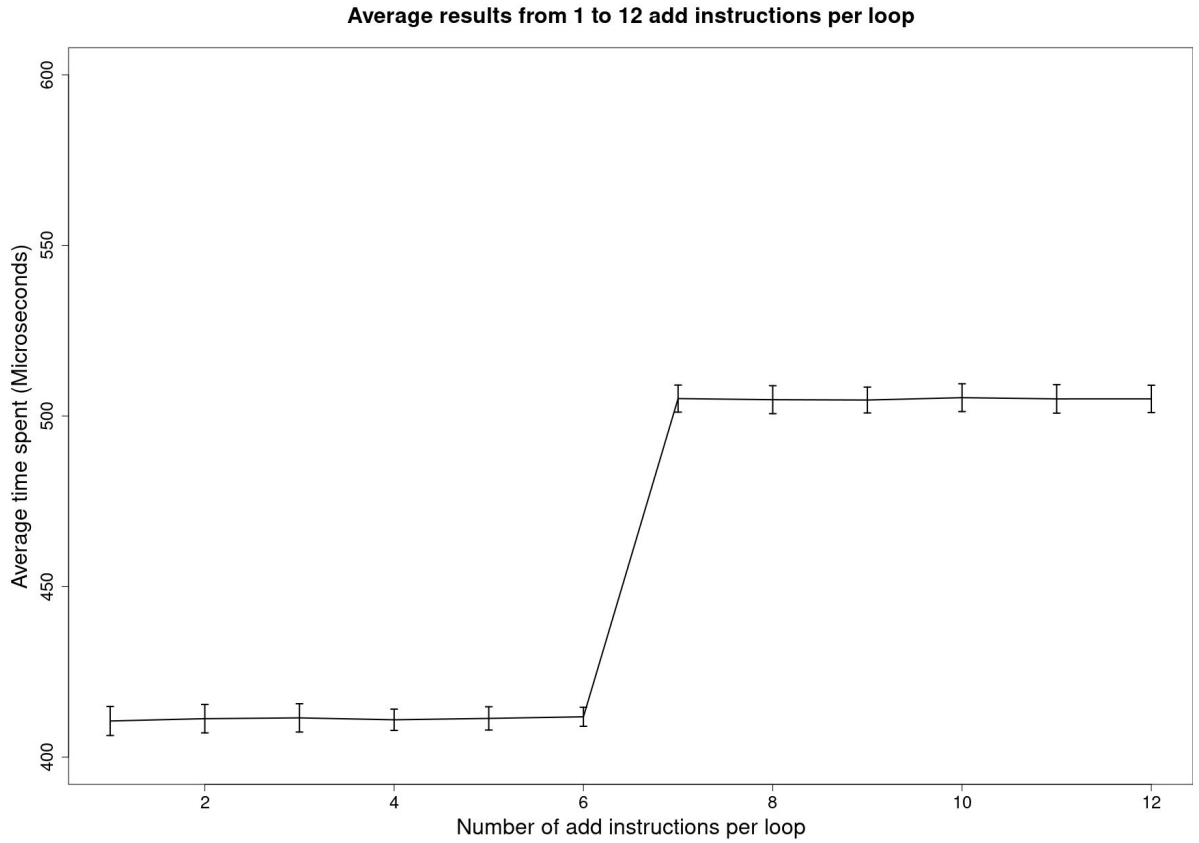
Xavier

- Number of cycles per second:
  - 2265MHz = 2.265*10^9
  - 1/(2.265*10^9) = 4.415*10^(-10)
- Number of NS per cycle
  - 4.42*10^(-10)/(10^(-9)) = 0.442ns
- Convert to MS
  - 0.442/1000 = 0.000442
- 1 Million cycles
  - 0.442*1000000 = 442 Microsec

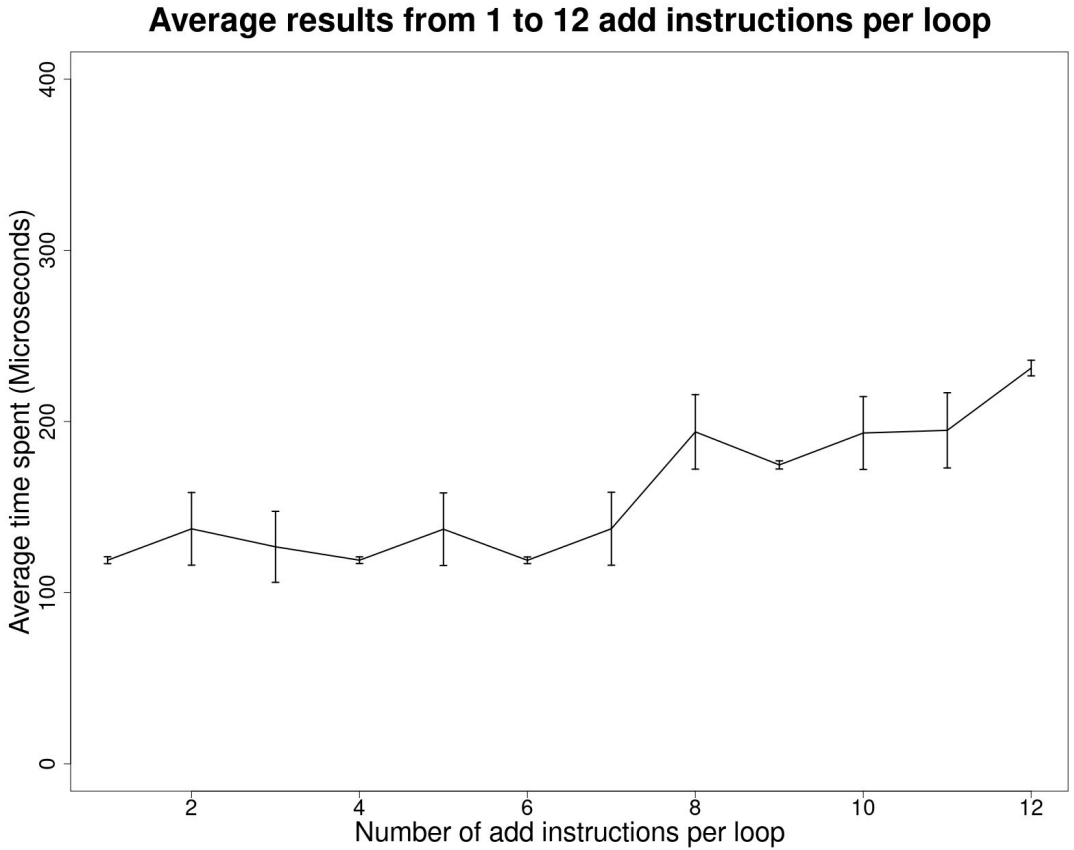
Results - Integer - TX2 A57



Results - Integer - TX2 Denver 2



Results - Integer - Xavier Carmel



How can we use FPUs effectively?

Hypothesis

- Multiple instructions in flight
- Should only need two
  - Only two FPUs

How do we measure it?

Calculating theoretical GFLOPS

$G = GHz \times Cores \times FPUs$

$G = 2.035 \times 1 \times 2 = 4.07$ (TX2)

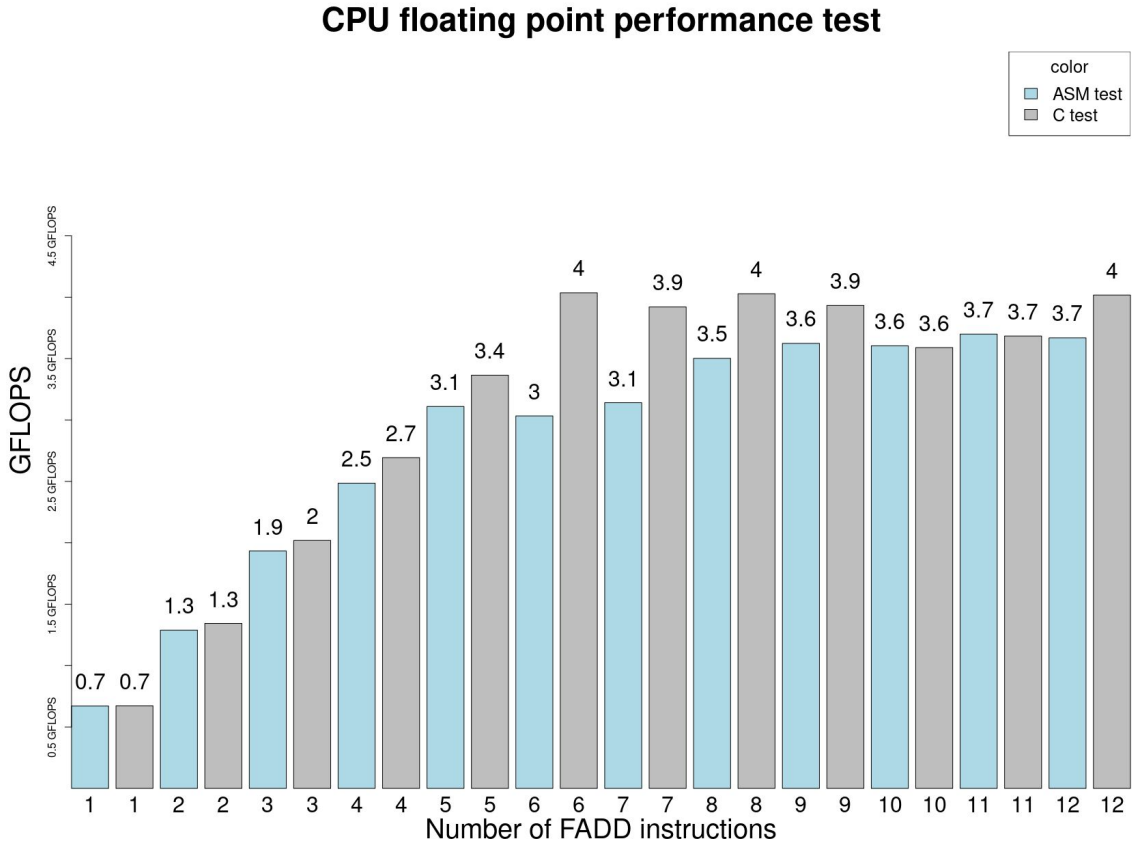
$G = 2.265 \times 1 \times 2 = 4.53$ (Xavier)

Calculating actual GFLOPS

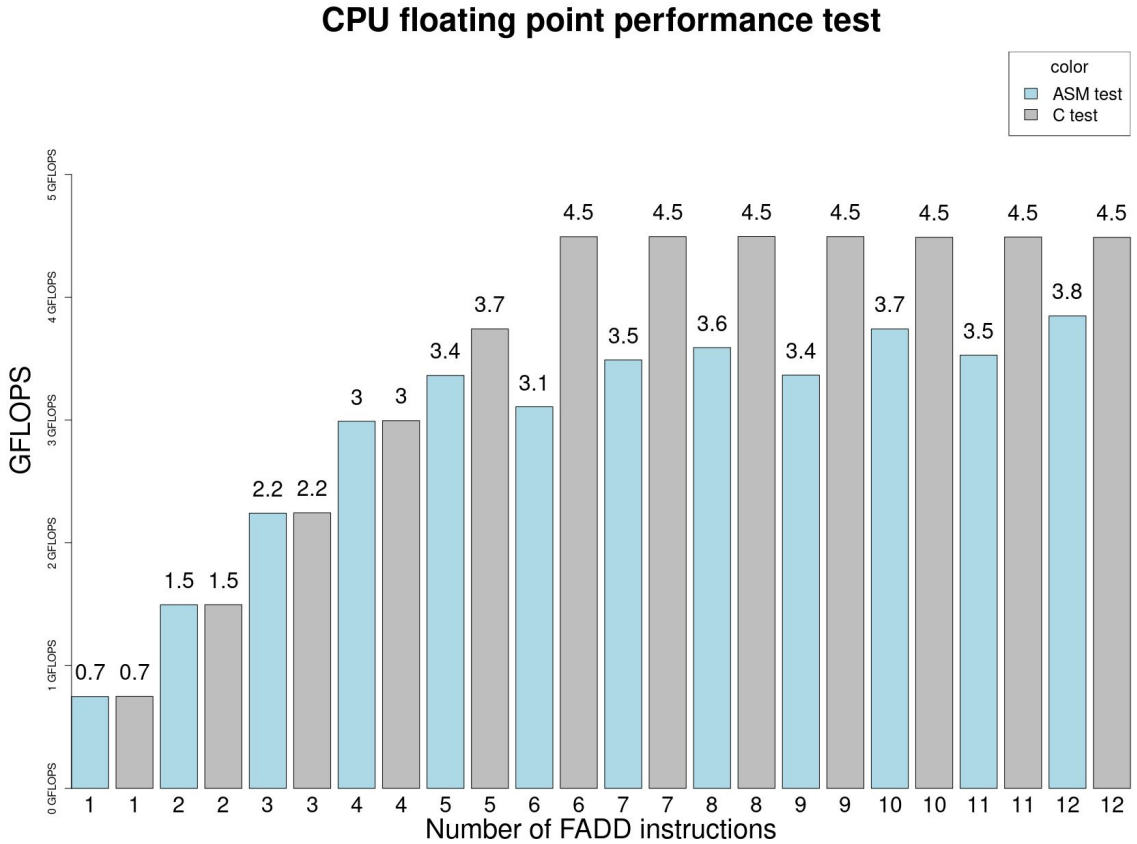
Flops = Ops / Seconds

G = Flops/10^9

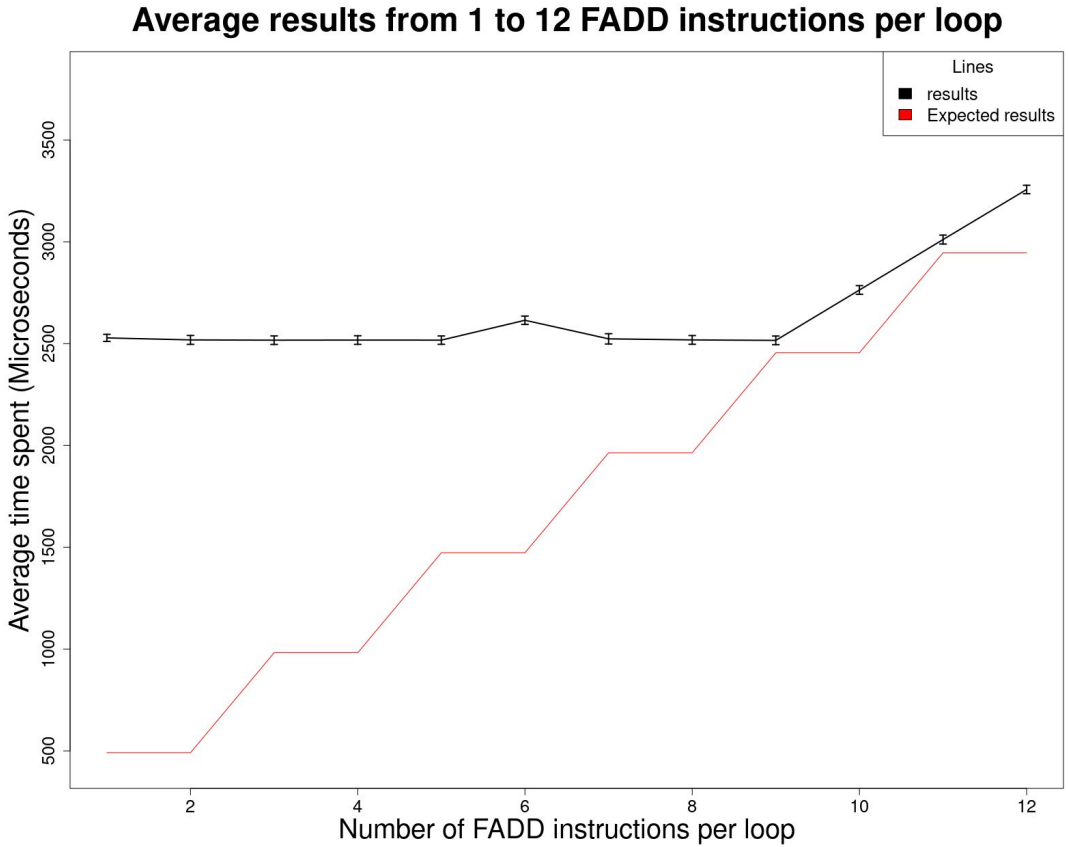
Case - FP add instructions - TX2



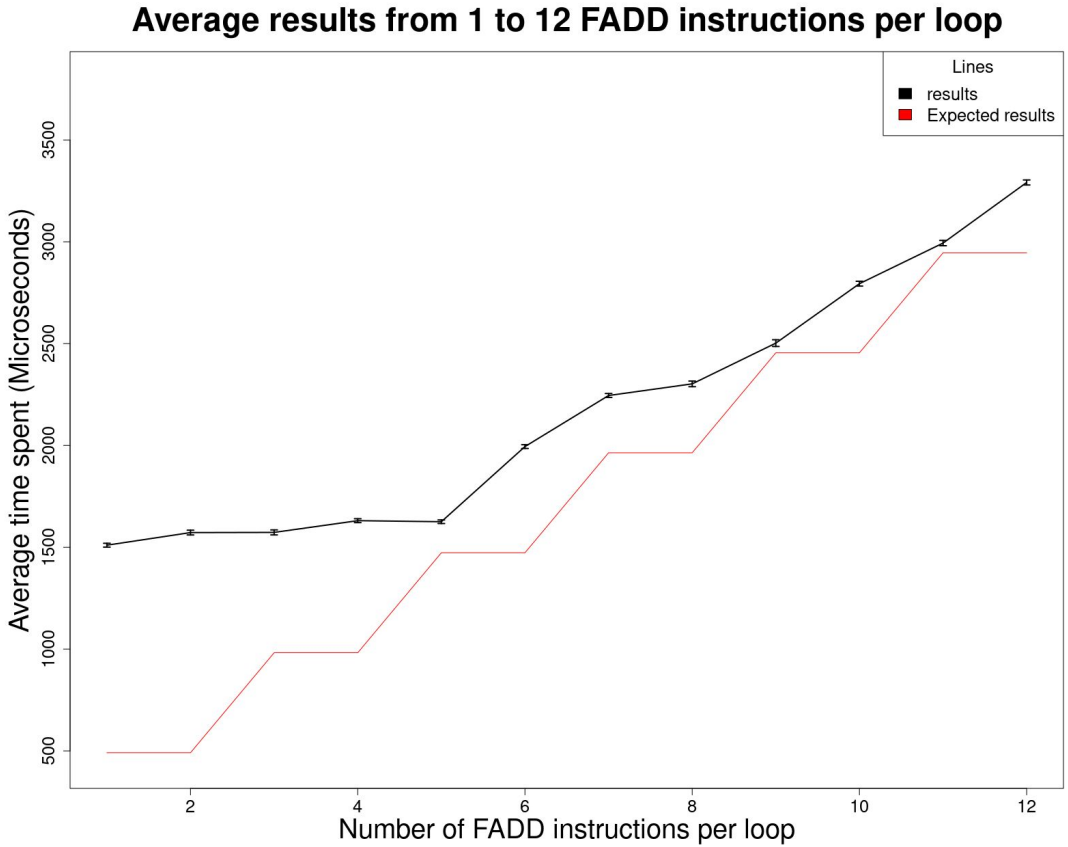
Case - FP add instructions - Xavier



Results - double - TX2 A57



Results - double - TX2 Denver 2

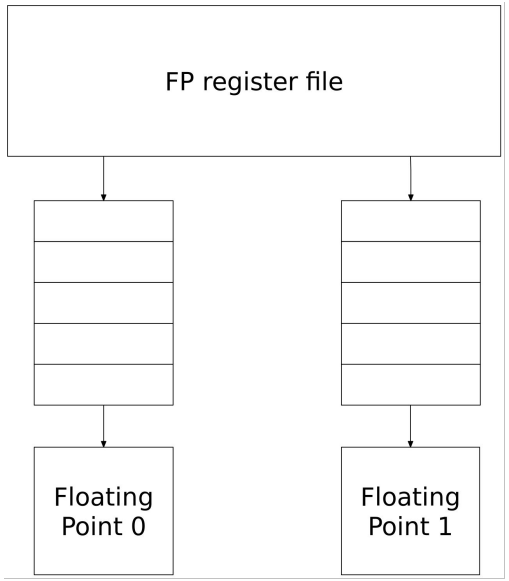


Why are the GFLOPS increasing?

- Any suggestions?

It’s all about the buffers and registers

- FP instructions are more complex than int instructions
- The CPU **can** perform fadd instructions in a single cycle
- **BUT**
  - You need to fill its buffer
  - The Xavier benefits from up to six instructions
  - More registers need to be used



How about NEON intrinsics?

- Was this the case with ARM NEON as well?
- Any guesses?

How do we measure it?

Calculating theoretical GFLOPS

G = GHz ✕ **Cores** ✕ **FPUs**

How do we measure it?

Calculating theoretical GFLOPS

G = GHz ✕ **Cores** ✕ **FPUs**

How do we measure it?

Calculating theoretical GFLOPS

G = GHz ✕ **Cores** ✕ **FPUs**

G = GHz ✕ Cores ✕ (FPUs ✕ Ops)

How do we measure it?

Calculating theoretical GFLOPS

$G = GHz \times \mathbf{Cores} \times \mathbf{FPUs}$

$G = GHz \times Cores \times (FPUs \times Ops)$

$G = 2.035 \times 1 \times (2 \times 4) = 16.28$

$G = 2.265 \times 1 \times (2 \times 4) = 18.12$

How do we measure it?

Calculating theoretical GFLOPS

$G = GHz \times \mathbf{Cores} \times \mathbf{FPUs}$

$G = GHz \times Cores \times (FPUs \times Ops)$

$G = 2.035 \times 1 \times (2 \times 4) = 16.28$

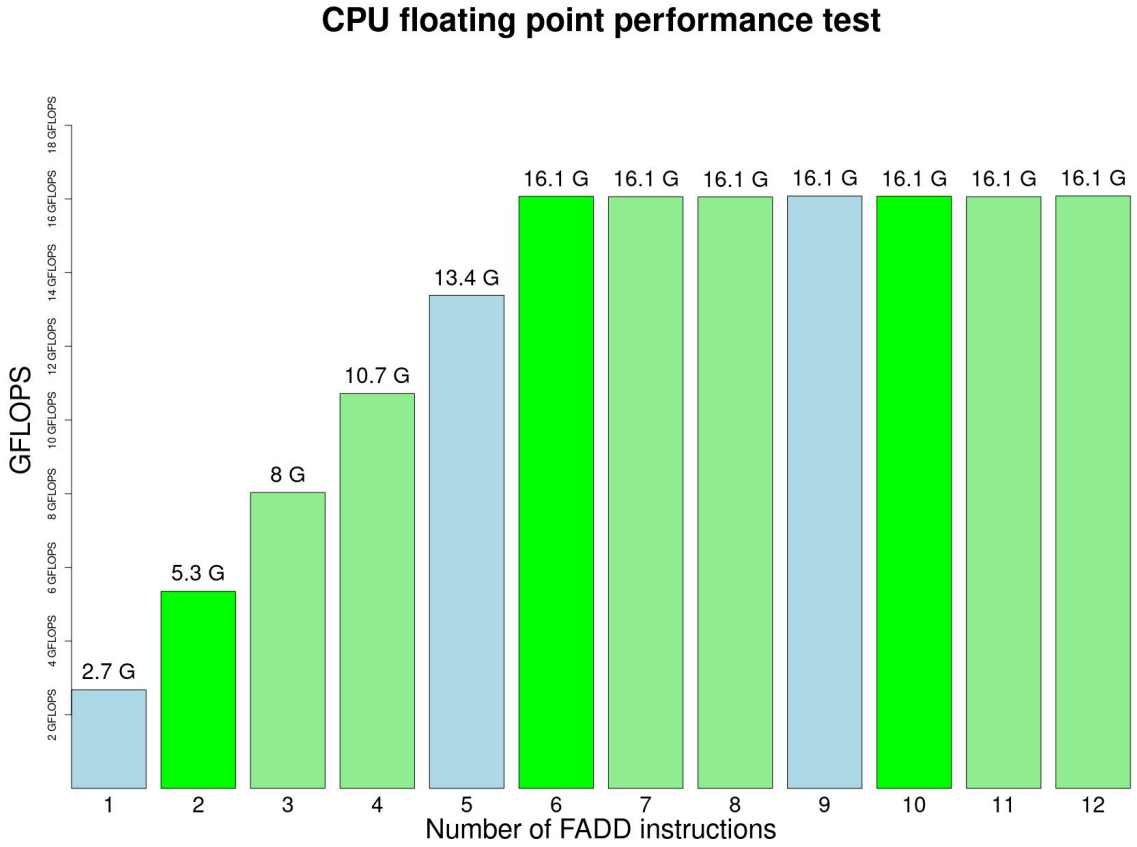
$G = 2.265 \times 1 \times (2 \times 4) = 18.12$

Calculating actual GFLOPS

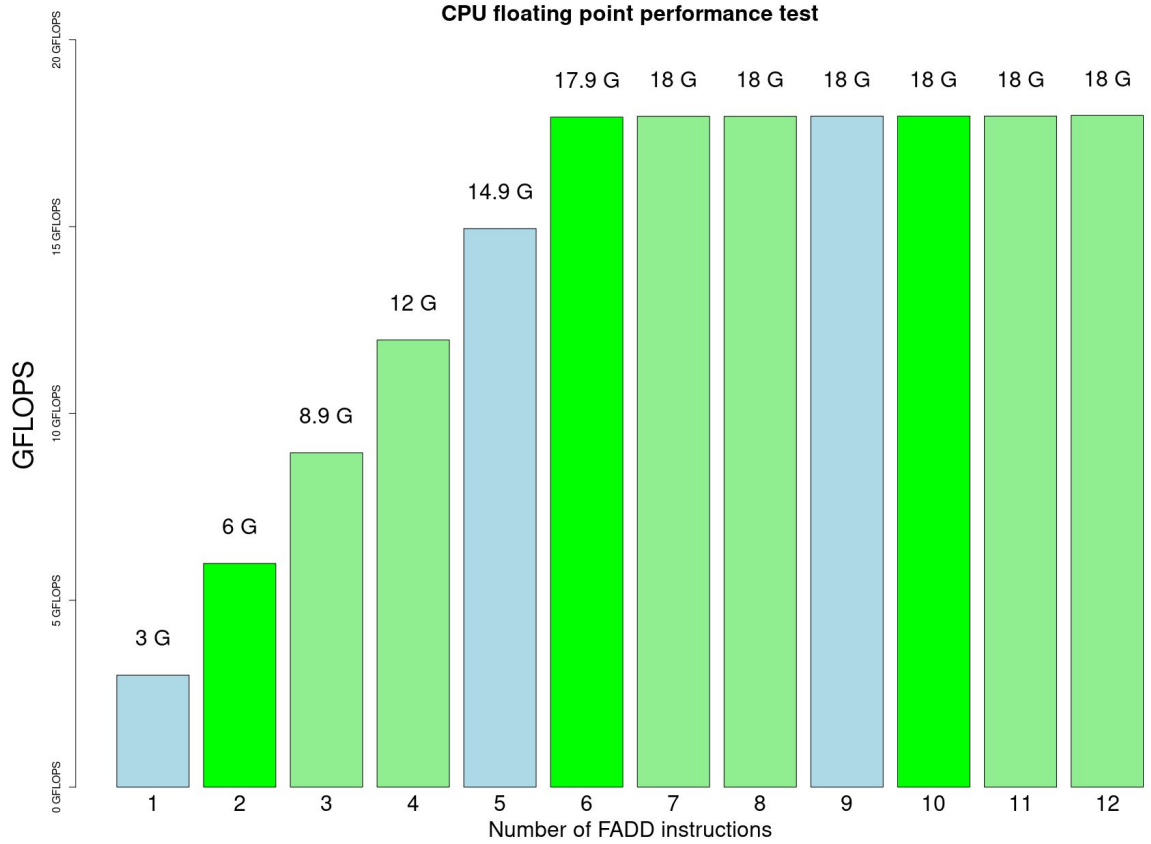
Flops = Ops / Seconds

G = Flops/10^9

Case - NEON ADD - TX2



Case - NEON ADD - Xavier



Dual-issued instructions

- FMA
  - Fused Multiply Accumulate
  - A single cycle

How do we measure that?

Calculating theoretical GFLOPS

$$G = GHz \times Cores \times (FPUs \times Ops)$$

$$G = 2.035 \times 1 \times (1 \times 8) = 16.28$$

$$G = 2.265 \times 1 \times (2 \times 8) = 36.24$$

Calculating actual GFLOPS

Flops = Ops / Seconds

G = Flops/10^9

How do we measure that?

Calculating theoretical GFLOPS

$G = GHz \times Cores \times (FPUs \times Ops)$

$G = 2.035 \times 1 \times (1 \times 8) = 16.28$

$G = 2.265 \times 1 \times (2 \times 8) = 36.24$

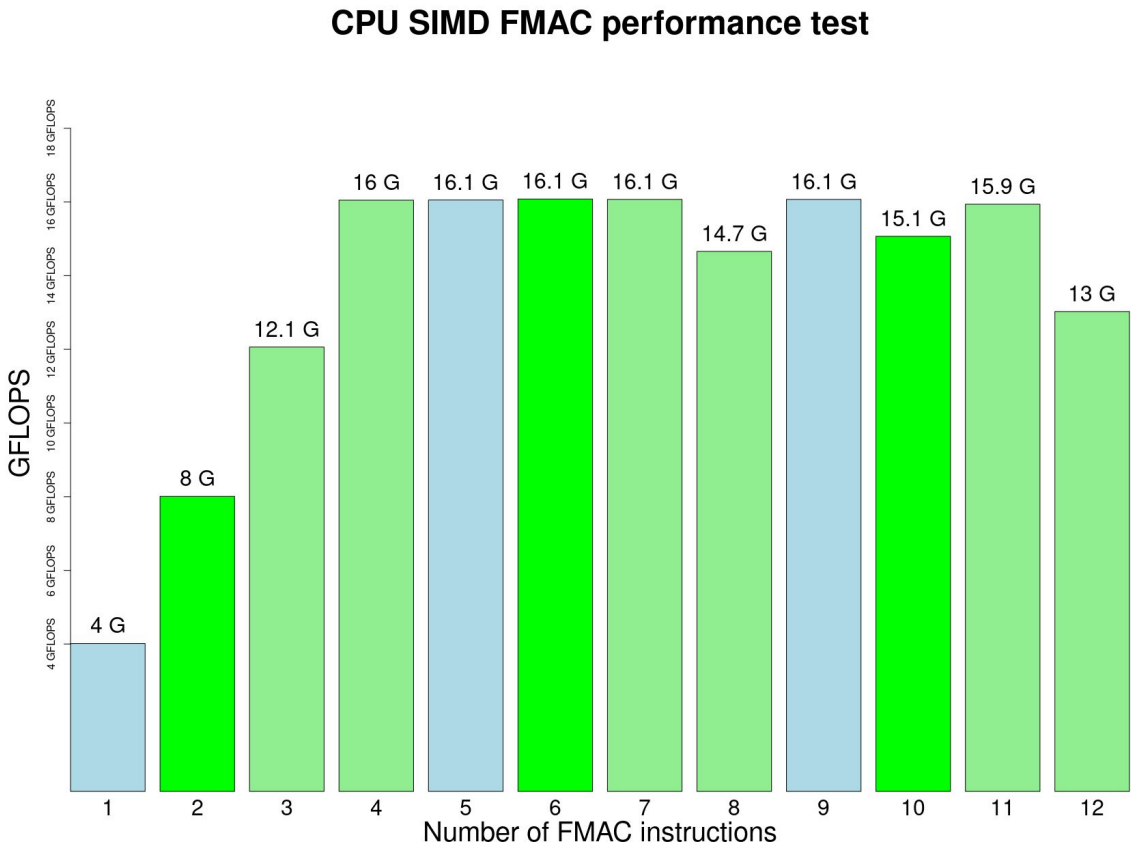
- Do we expect anything to change?

Calculating actual GFLOPS

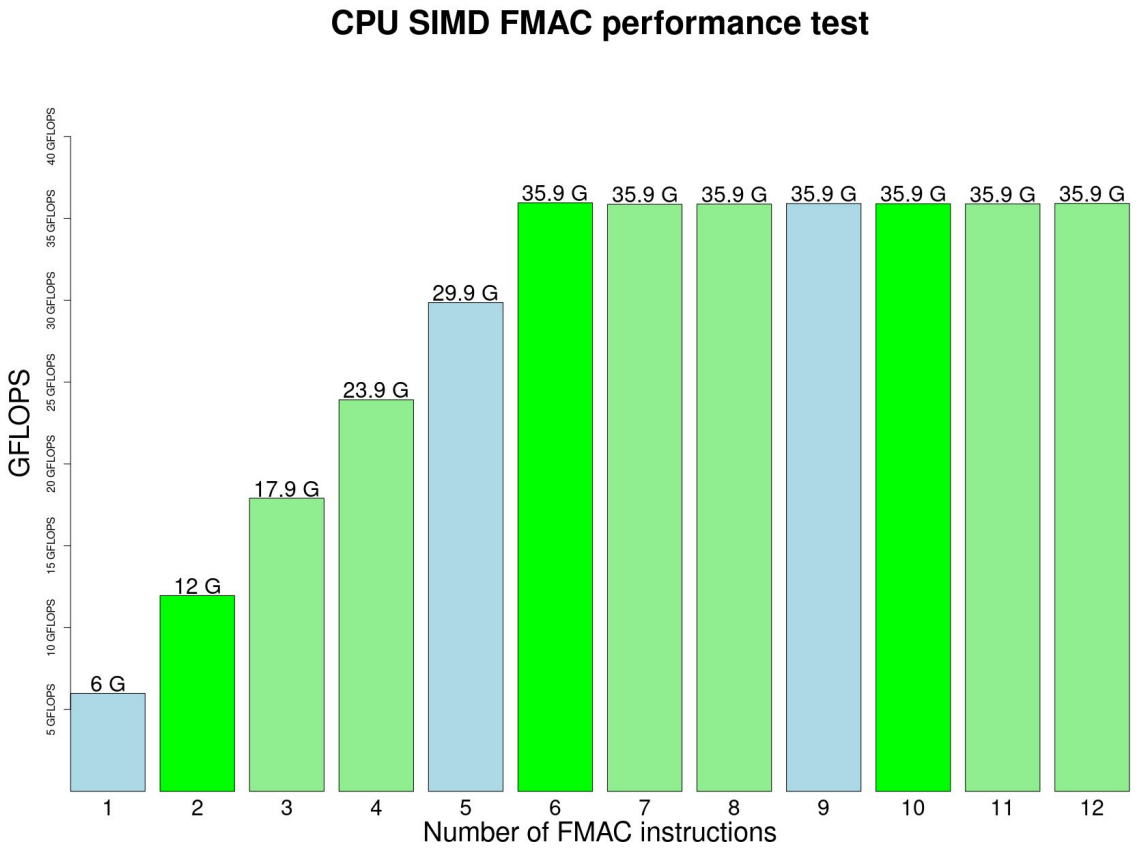
Flops = Ops / Seconds

G = Flops/10^9

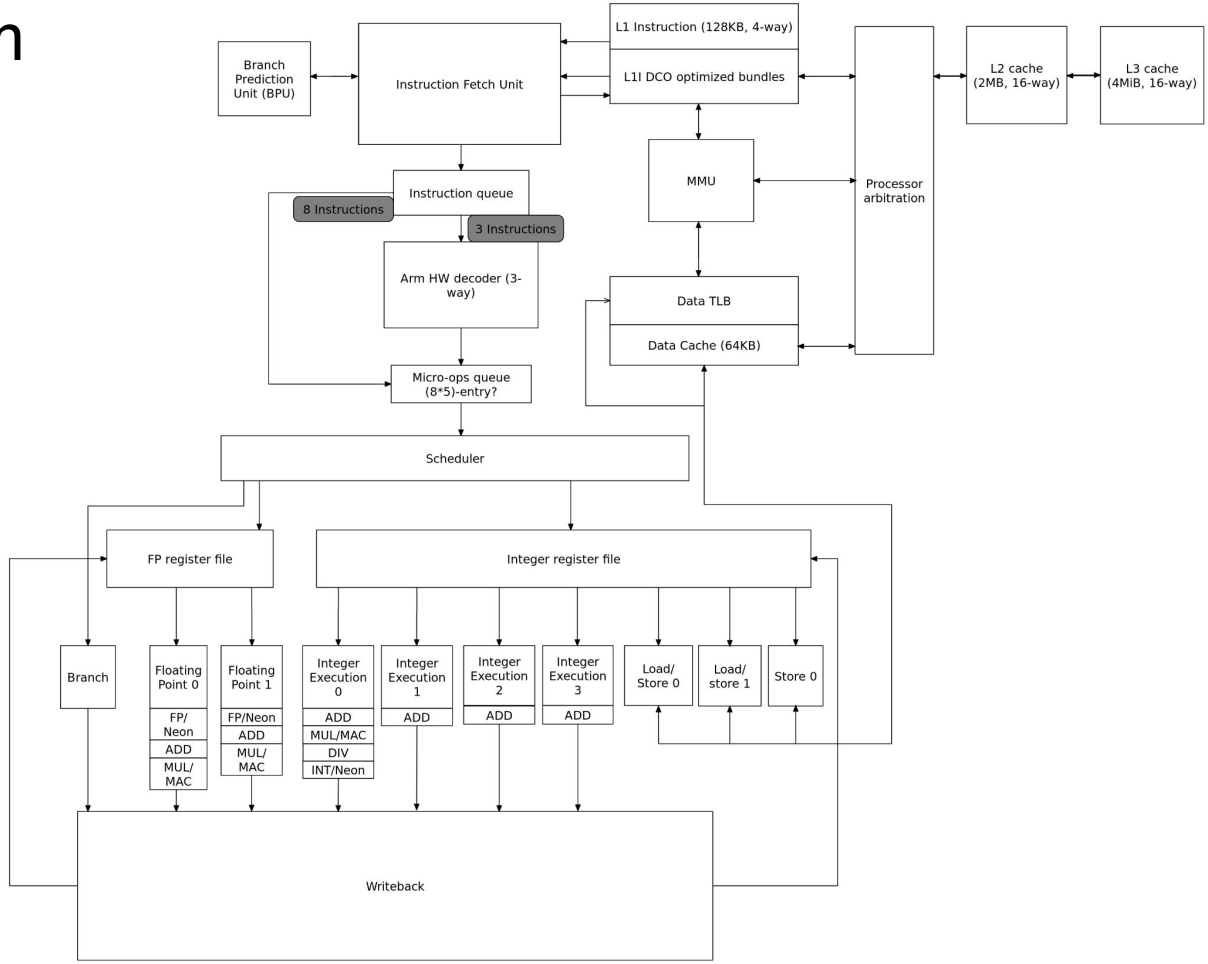
Results - GFLOPS - TX2 - Denver 2



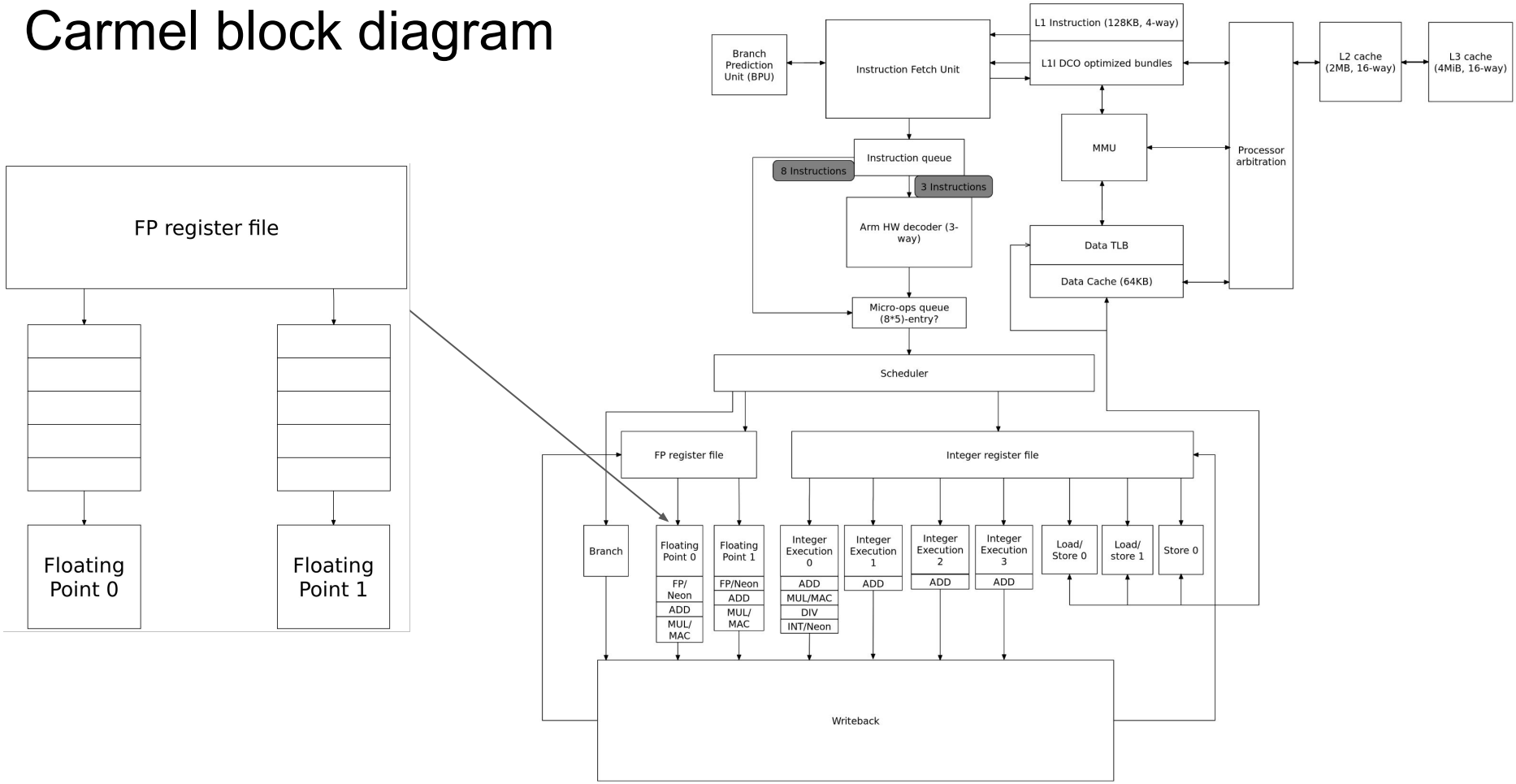
Results - GFLOPS - Xavier - Carmel



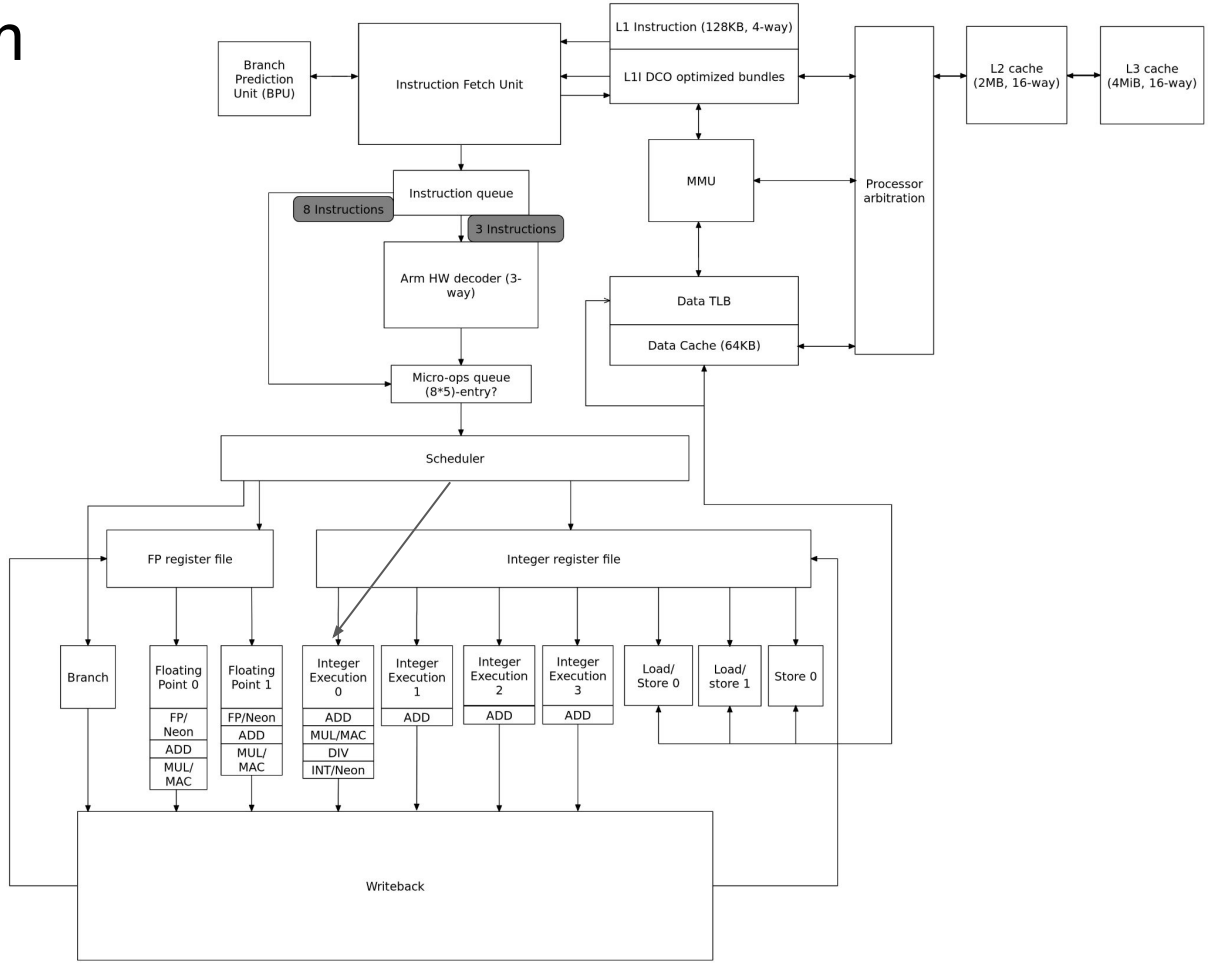
Carmel block diagram



Carmel block diagram



Carmel block diagram



Conclusion

- CPU ALUs
  - Made for general purpose use
  - Not perfect for heavy compute
  - SIMD can help there!
- Xavier tips and tricks
  - Keep thinking parallel
    - You still have two FPUs, not just one!
  - Multiple instructions in flight may be good!
    - Use more registers!

Thank you!

Any questions?