ASIC accelerators

To read more...

This day's papers:

- Reagan et al, "Minerva: Enabling Low-Power, Highly-Accurate Deep Neural Network Accelerators"
- Shao et al, "The Aladdin Approach to Accelerator Design and Modeling" (Computer magazine version)

Supplementary reading:

- Han et al, "EIE: Efficient Inference Engine on Compressed Neural Networks"
- Shao et al, "Aladdin: A Pre-RTL, Power-Performance Accelerator Simulator Enabling Large Design Space Exploration of Customized Architectures"

A Note on Quoting Papers

I didn't look closely enough at paper reviews earlier in the semester

Some paper reviews copying phrases from papers

You must make it obvious you are doing so

This will get you in tons of trouble later if you don't have good habits

Usually — better off rewriting completely

even if your grammar is poor

Consistent style — easier to read

Homework 3 Questions?

Part 1 — due tomorrow, 11:59PM

Part 2 — serial codes out

Accelerator motivation

end of transistor scaling

specialization as way to further improve performance

especially performance per watt

key challenge: how do we design/test custom chips quickly?

Behavioral High-Level Synthesis

take C-like code, produce HW

problem (according to Aladdin paper):

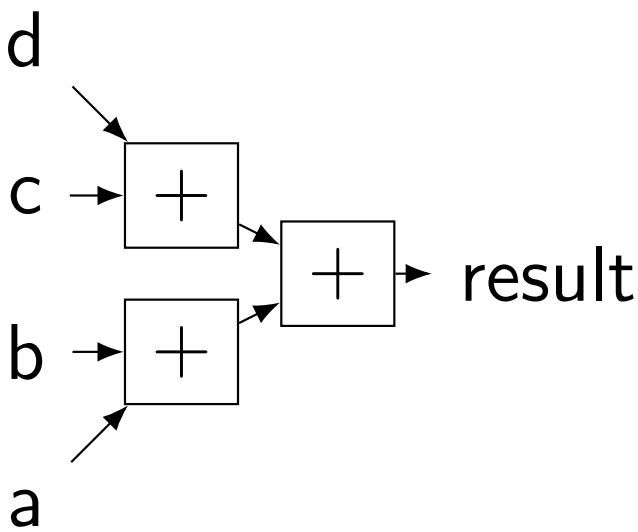
requires lots of tuning...

to handle/eliminate dependencies

to make memory accesses/etc. efficient

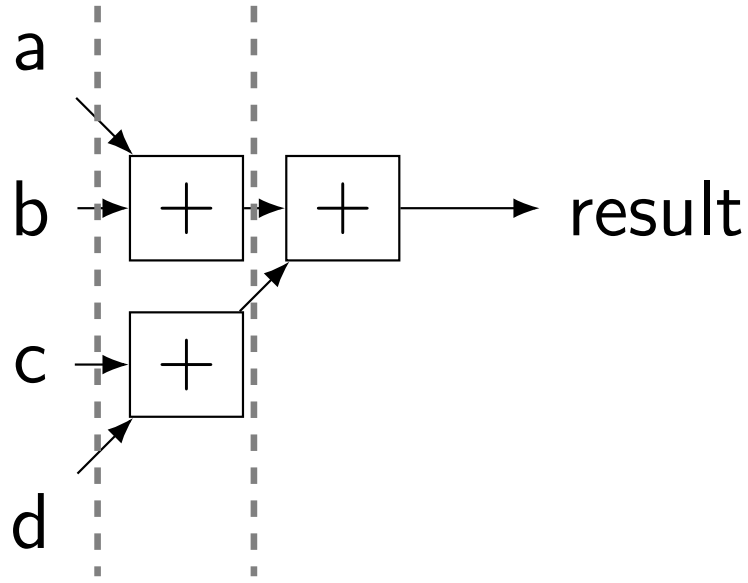
Data Flow Graphs

```
int sum_ab = a + b;
int sum_cd = c + d;
int result = sum_ab + sum_cd;
```

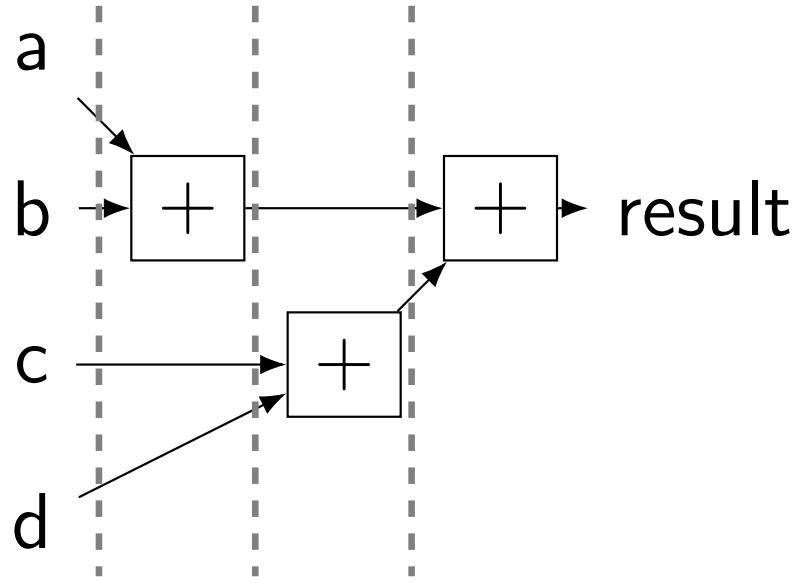


DFG scheduling

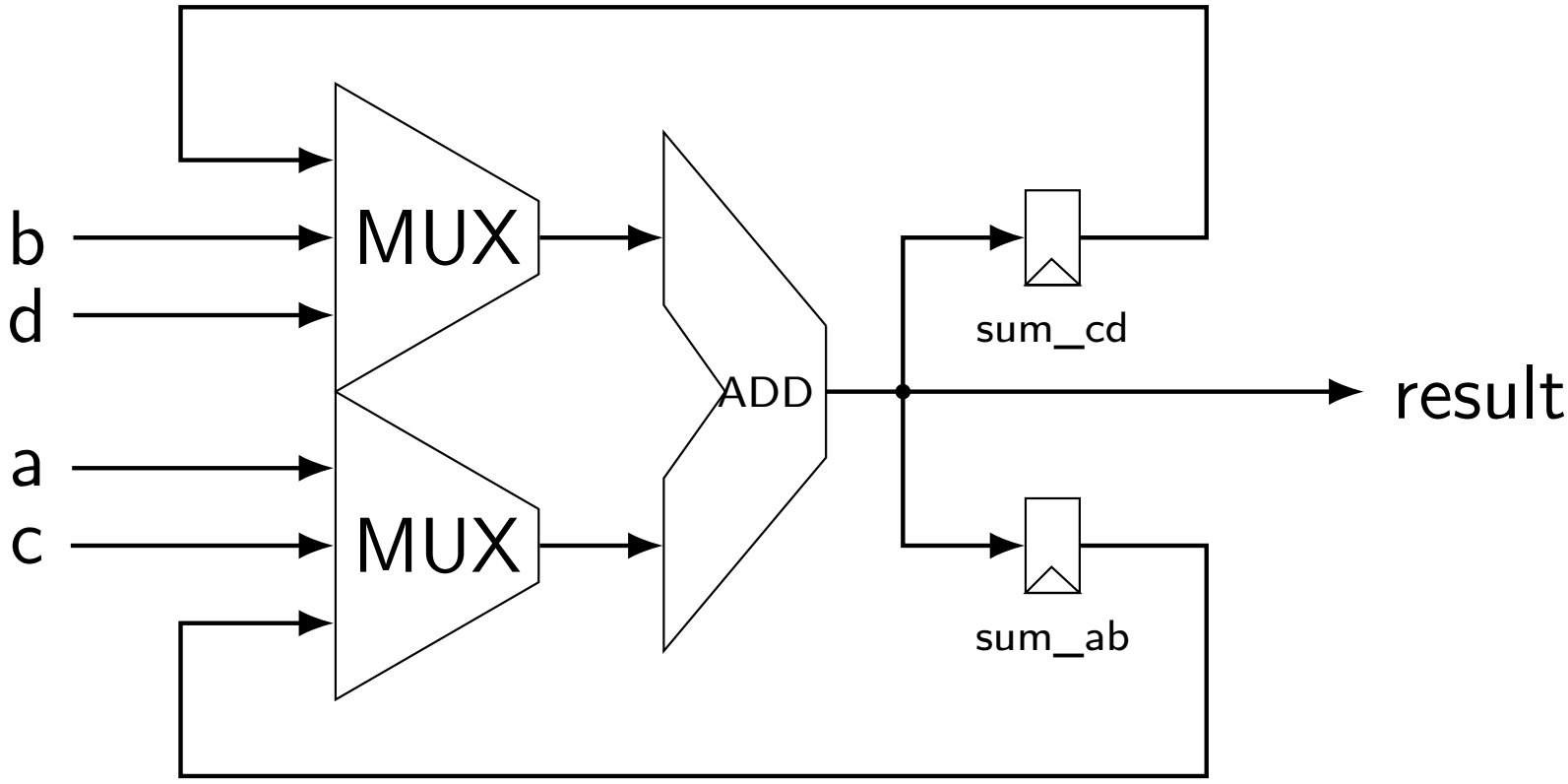
two add functional units:



one add functional unit:



DFG realization — data path



plus control logic

selectors for MUXes, write enable for regs

Dynamic DDG

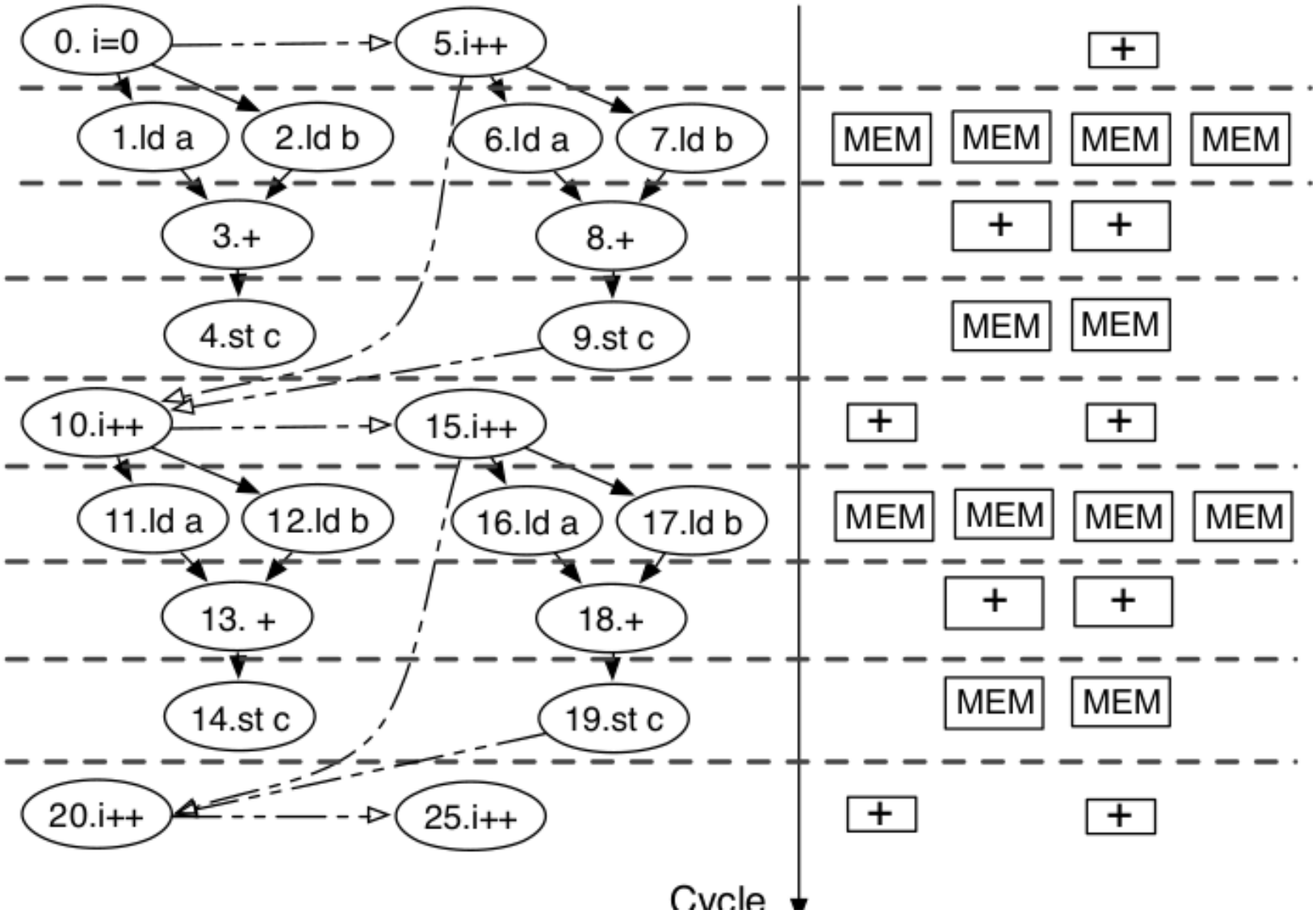
Aladdin trick:

use dynamic (runtime) dependencies
assume someone will figure out scheduling HW

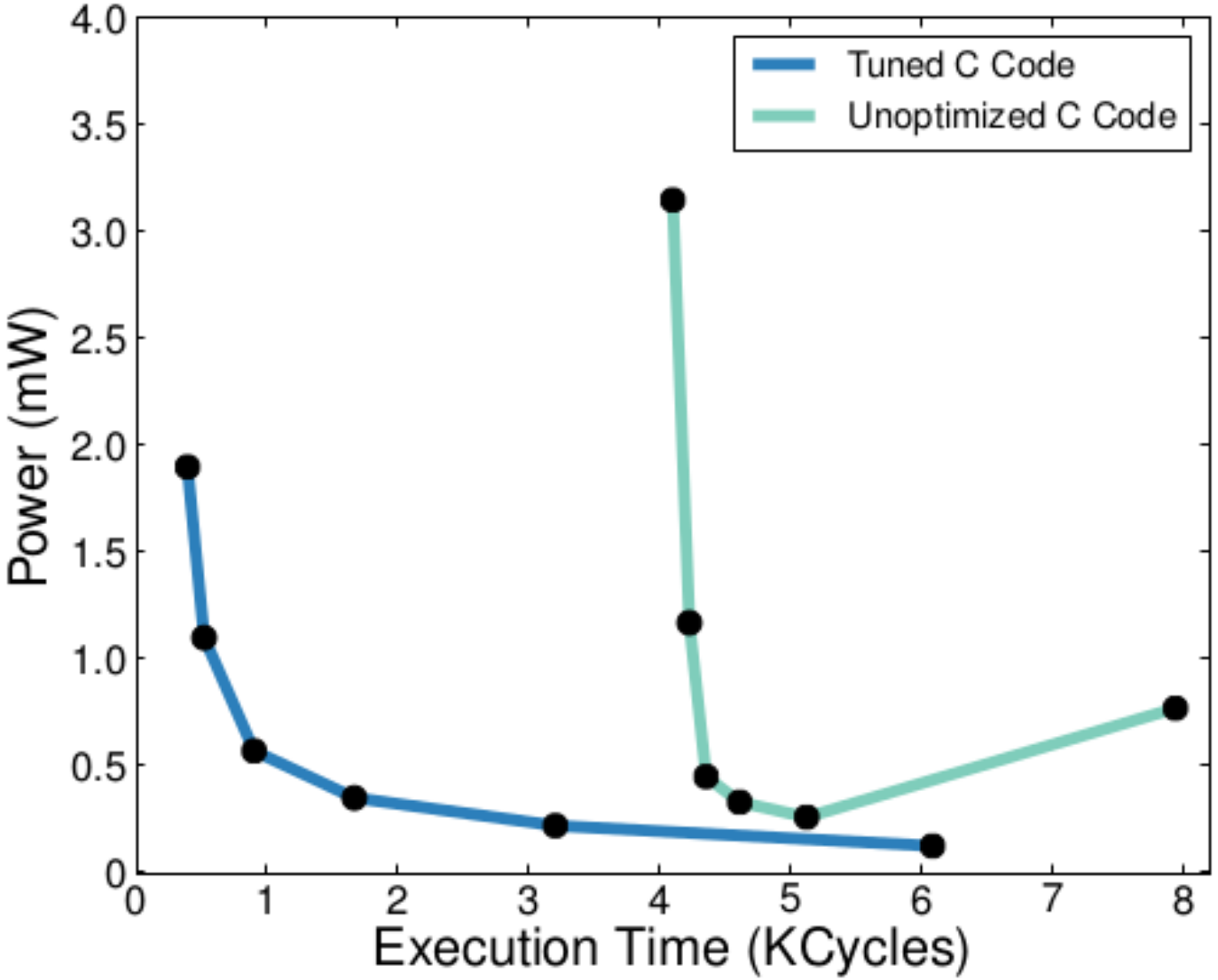
full synthesis:

actually need to make working control logic
need to figure out memory/register connections

Dynamic Data Dependency Graph



full synthesis: tuning



tuning: false dependencies

"the reason is that when striding over a partitioned array being read from and written to in the same cycle, though accessing different elements of the array, the HLS compiler conservatively adds loop-carried dependences."

Aladdin area/power modeling

functional unit power/area + memory power/area

library of functional units

- tested via microbenchmarks

memory model

- select latency, number of ports (read/write units)

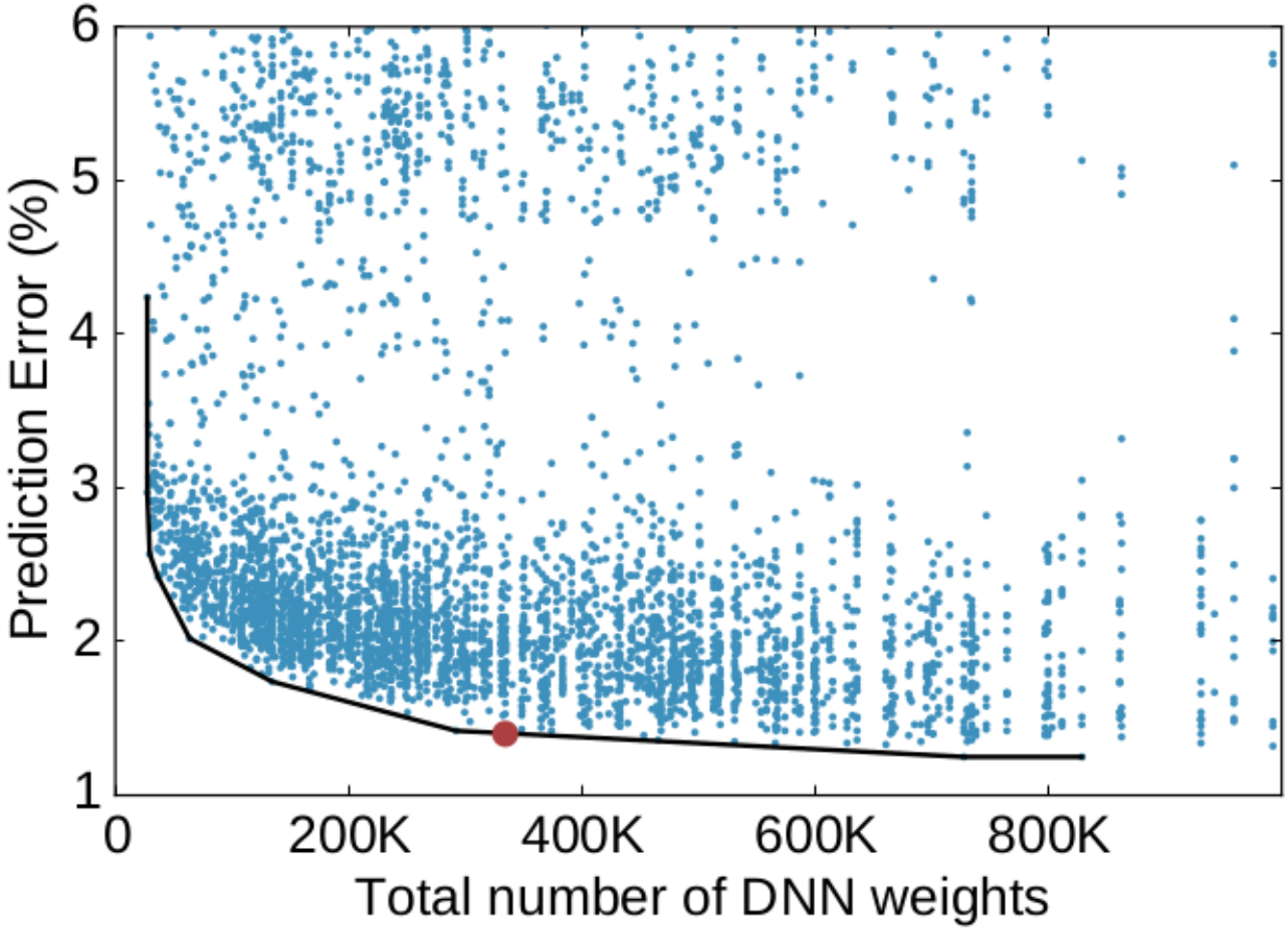
Missing area/power modeling

control logic accounting

wire lengths, etc., etc.

Pareto-optimum

Pareto-optimum: can't make anything better without making something worse



design space example (GEMM)

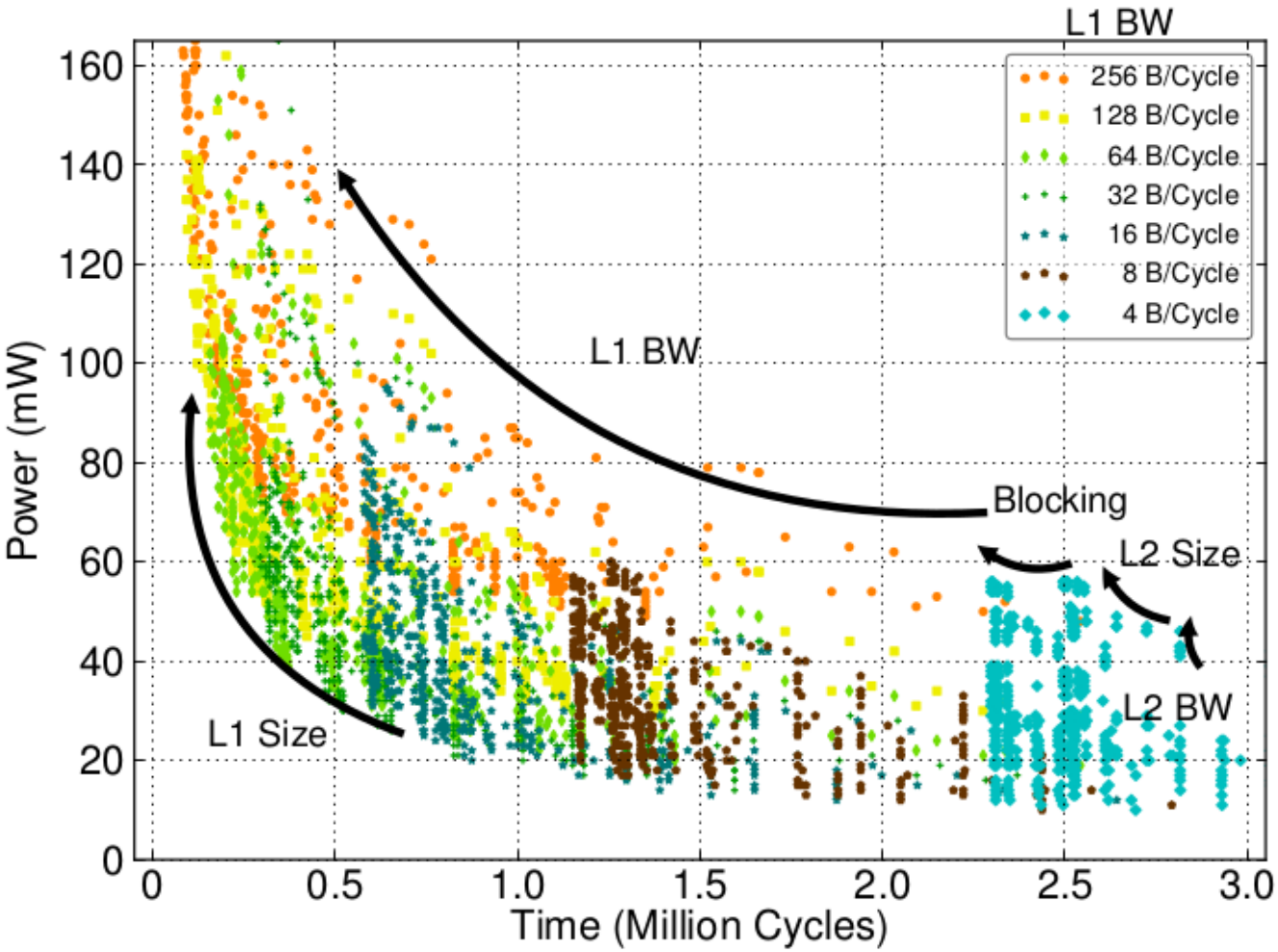
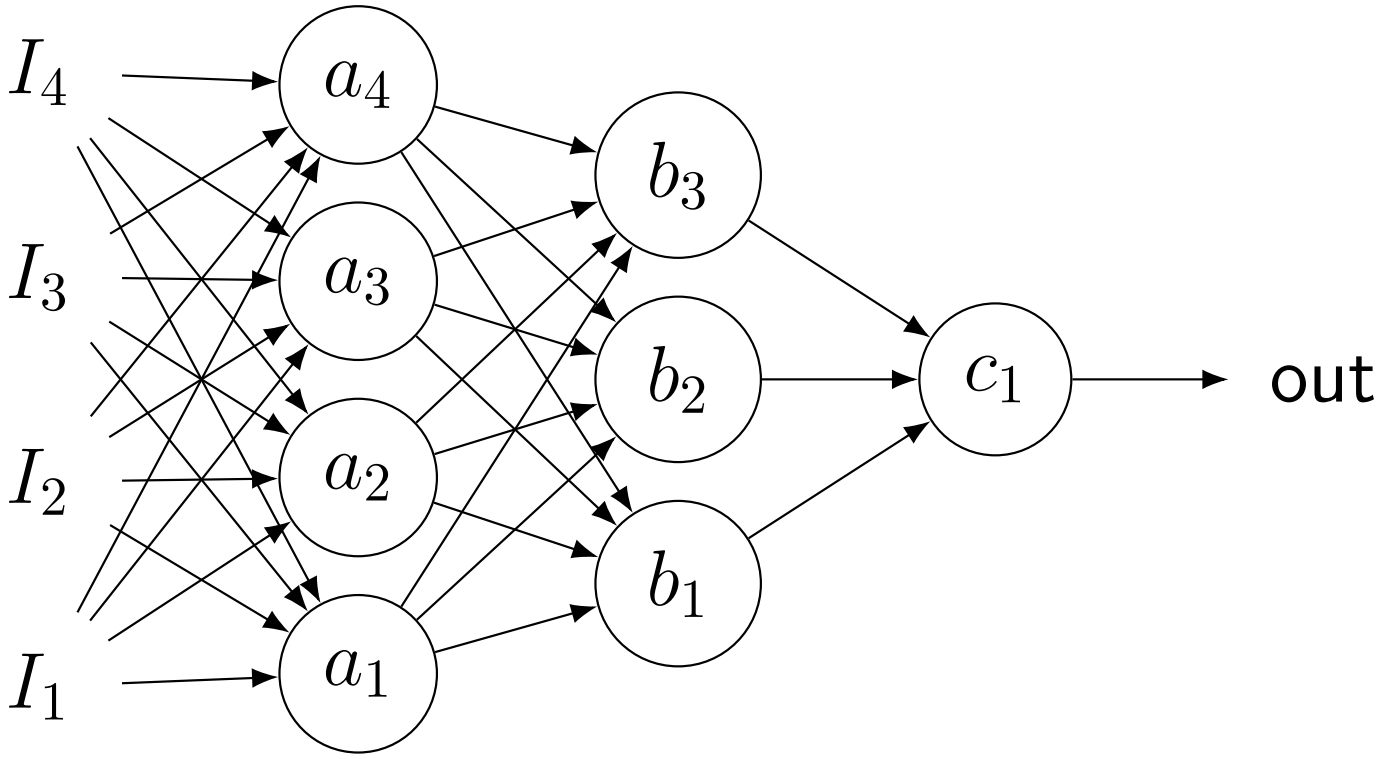


Figure 11: GEMM design space.

Neural Networks (1)

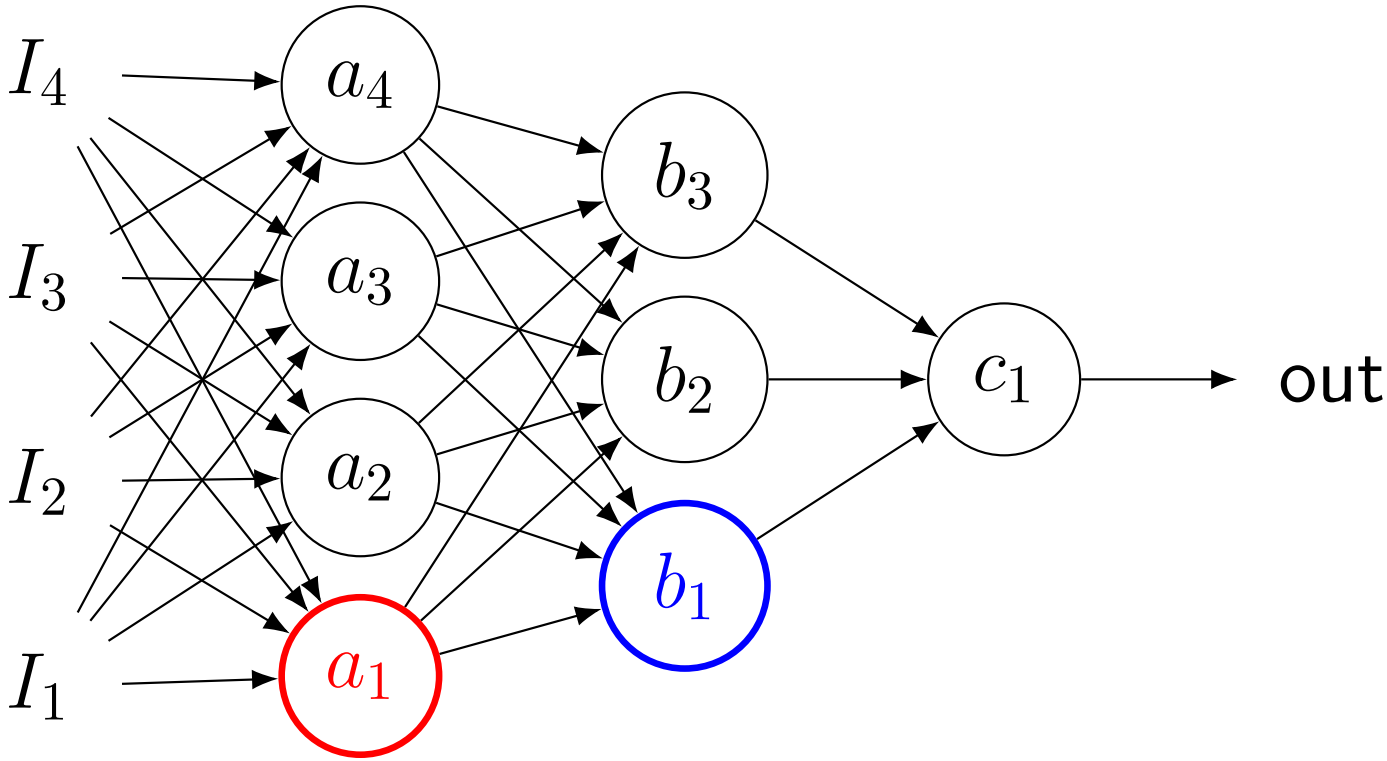


real world: $out_{real} = F(I_1, I_2, I_3, I_4)$

compute approximation $out_{pred} \approx \hat{F}(I_1, I_2, I_3, I_4)$

using intermediate values a_is, b_is

Neural Networks (2)



$$a_1 = K\left(w_{a_1,1}I_1 + w_{a_1,2}I_2 + \cdots + w_{a_1,4}I_4\right)$$

$$b_1 = K\left(w_{b_1,1}a_1 + w_{b_1,2}a_2 + w_{b_1,3}a_3\right)$$

ws — weights, selected by training

Neural Networks (3)

neuron: $a_1 = K\left(w_{a_1,1}I_1 + w_{a_1,2}I_2 + \cdots + w_{a_1,4}I_4\right)$

$K(x)$ — activation function, e.g. $\dfrac{1}{1+e^{-x}}$

- close to 0 as x approaches $-\infty$
- close to 1 as x approaches $+\infty$
- differentiable

Minerva's problem

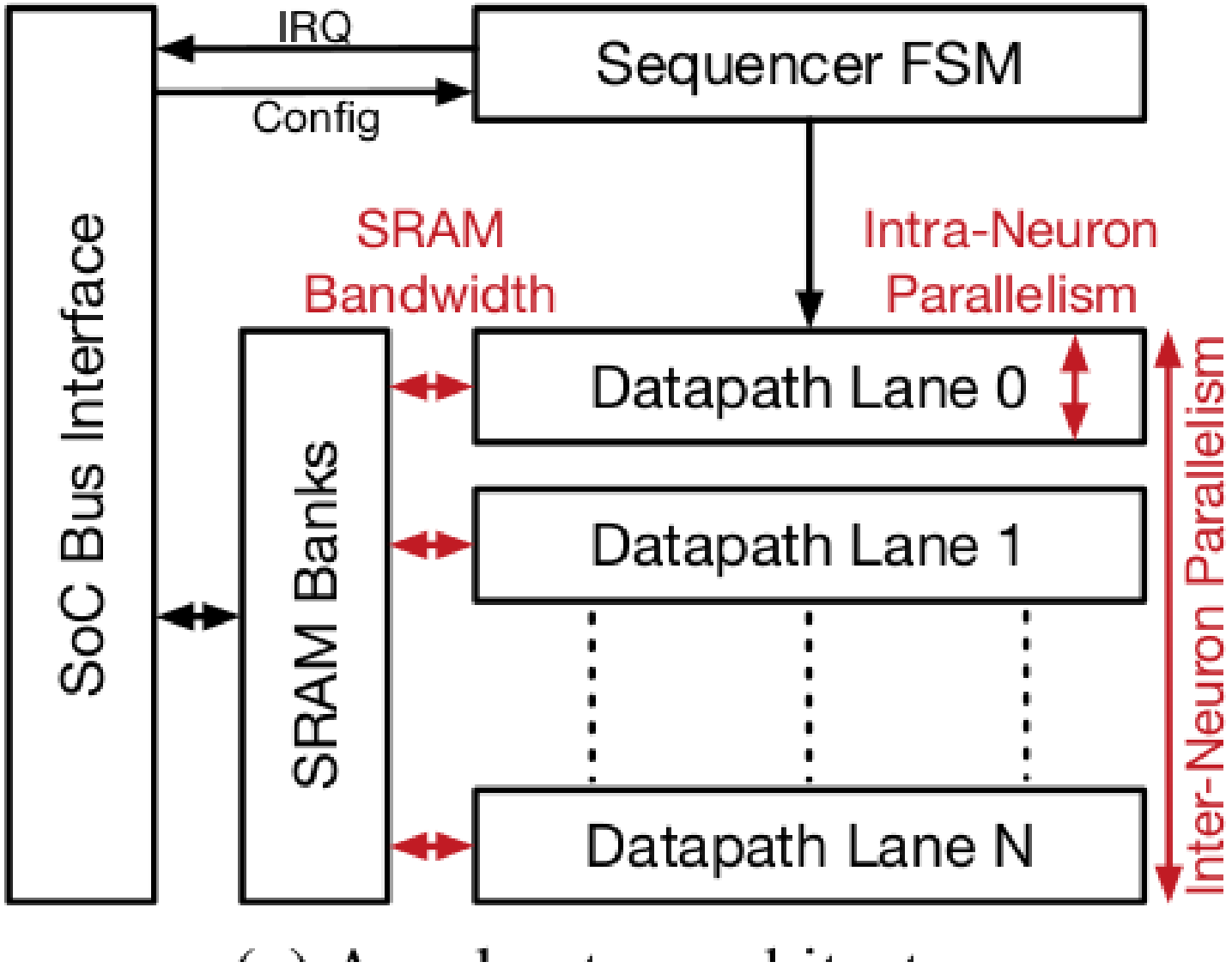
evaluating neural networks

train model once, deploy in portable devices

example: handwriting recognizer

goal: low-power, low-cost ($\approx$ area) ASIC

High-level design



Tradeoffs

mathematical — design of neural network

- hardware — size of memory, number of calculations

mathematical — precision of calculations

- hardware — size of memory, number of calculations

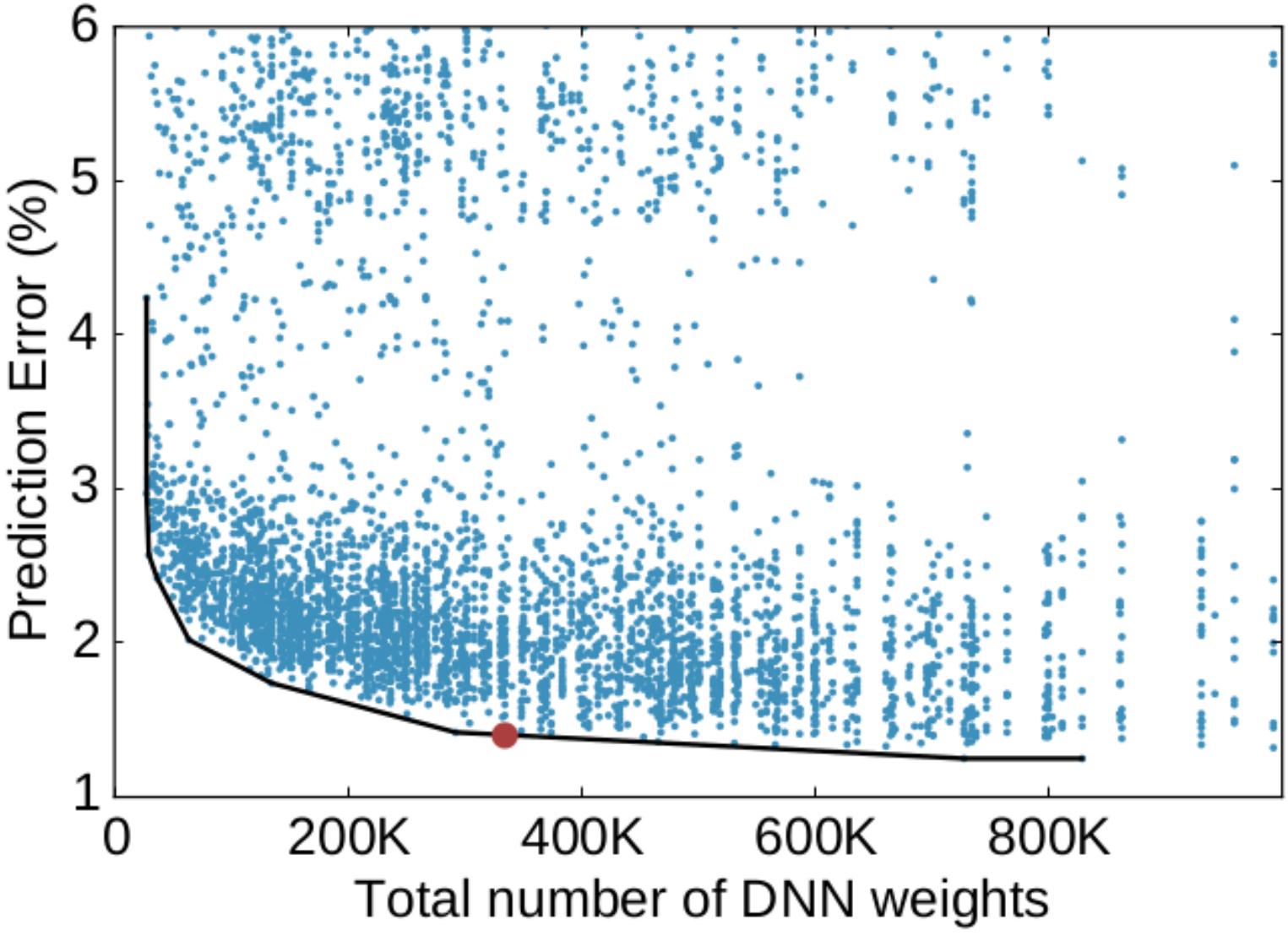
hardware — amount of inter-neuron parallelism

- approx. cores

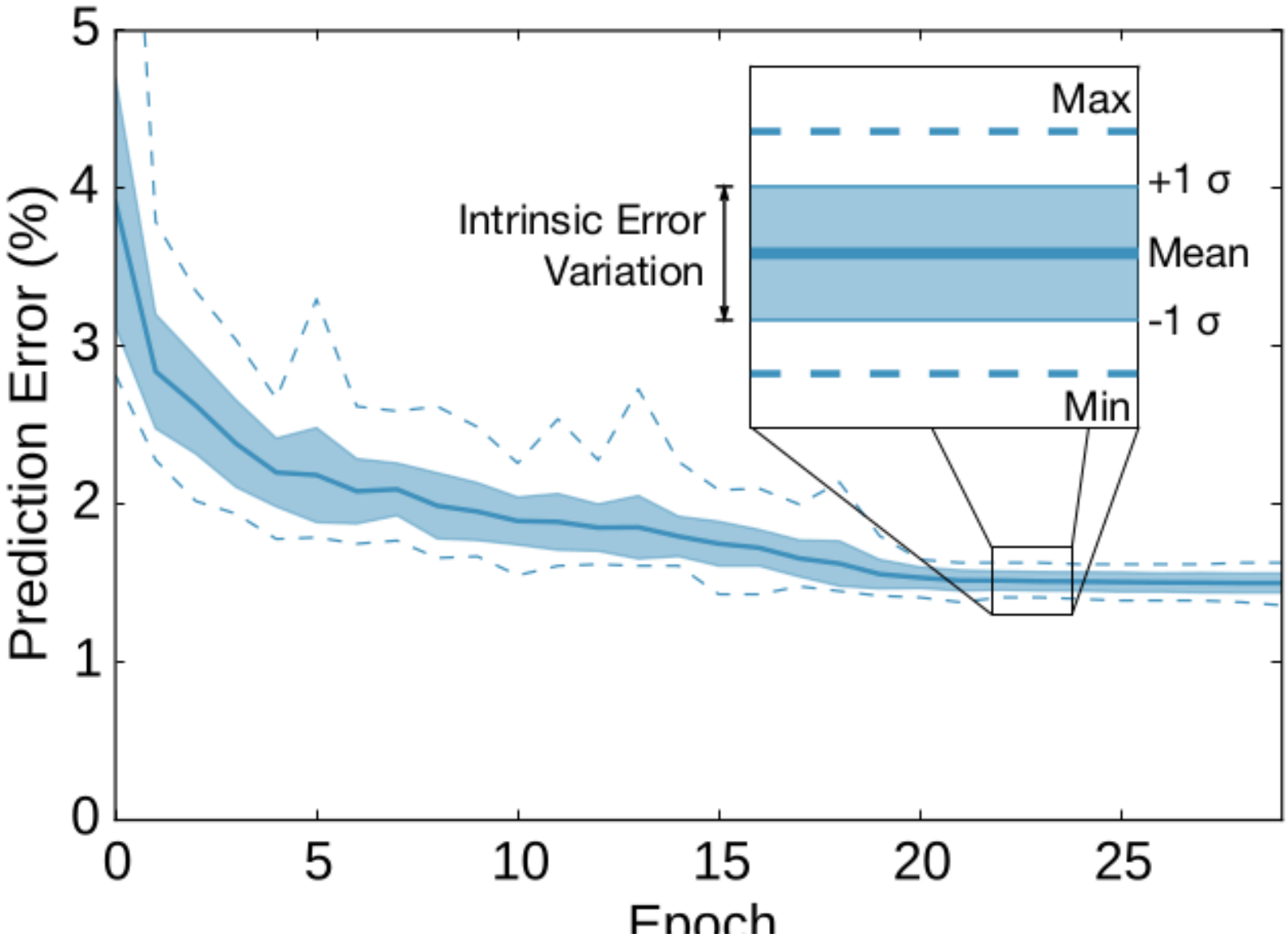
hardware — amount of intra-neuron parallelism

- i.e. pipeline depth

Neural network parameters



“intrinsic inaccuracy”

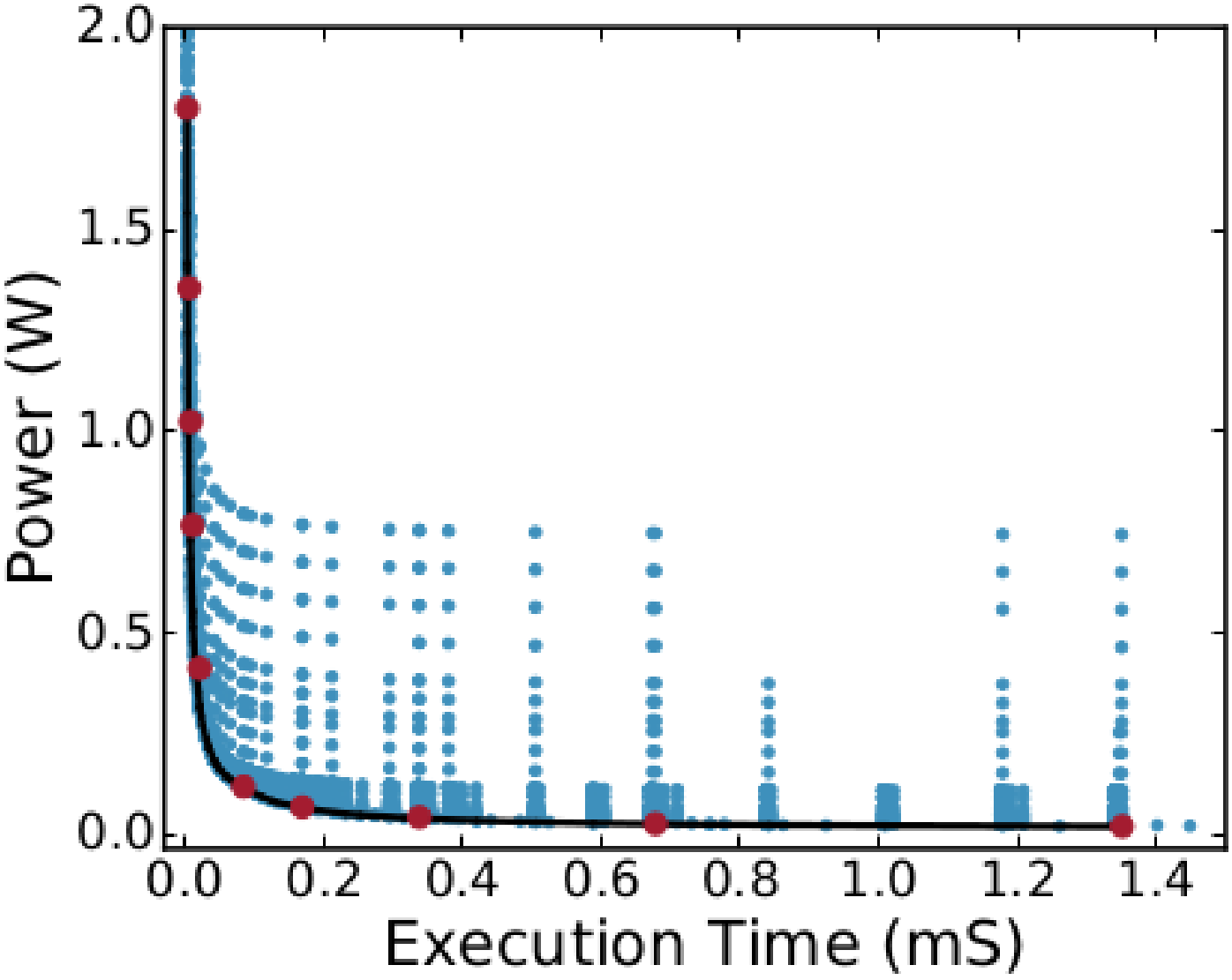


intrinsic inaccuracy assumption

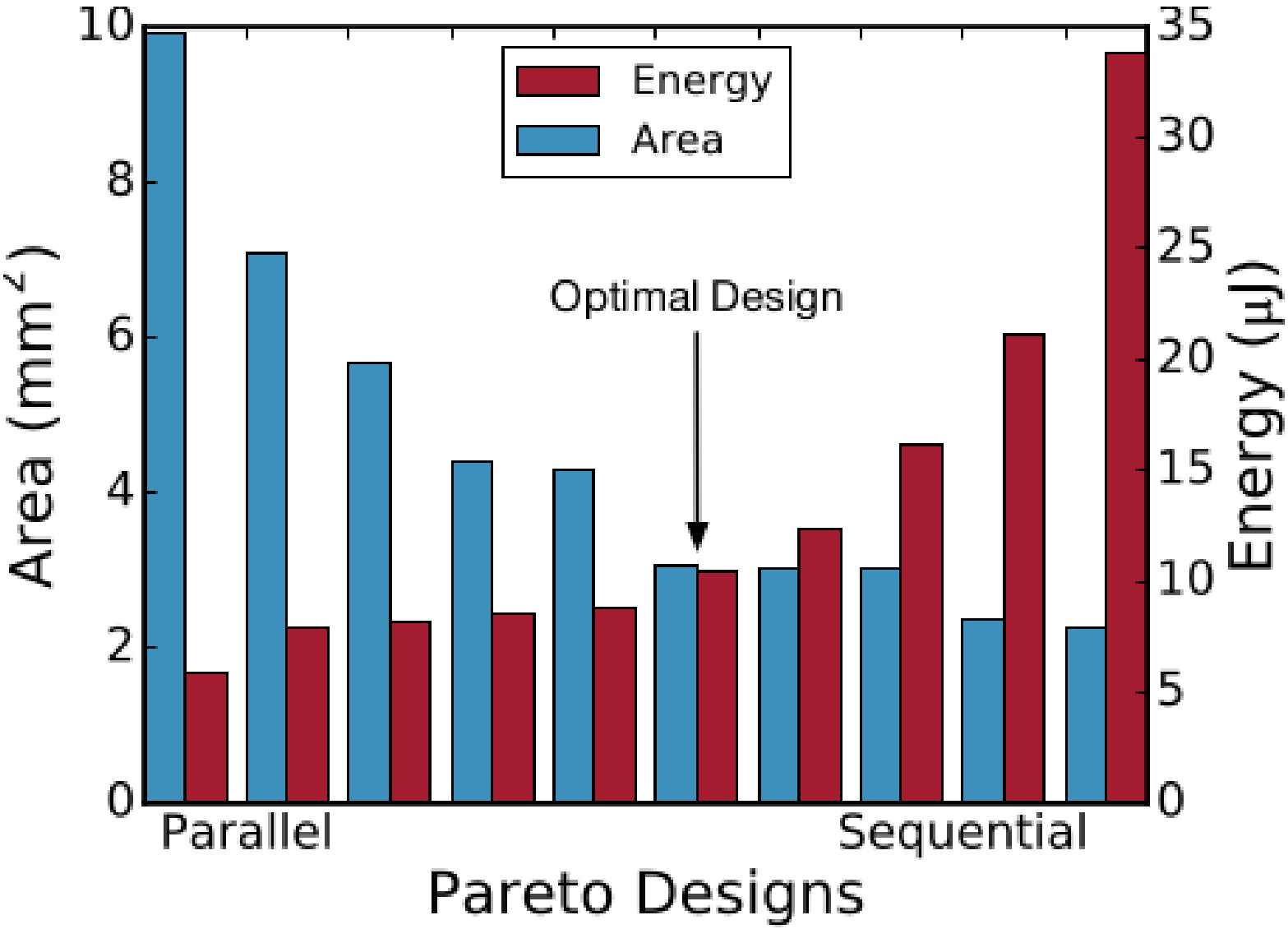
don't care if precision variation similar to training variation

sensible?

HW tradeoffs (1)



HW tradeoffs (1)

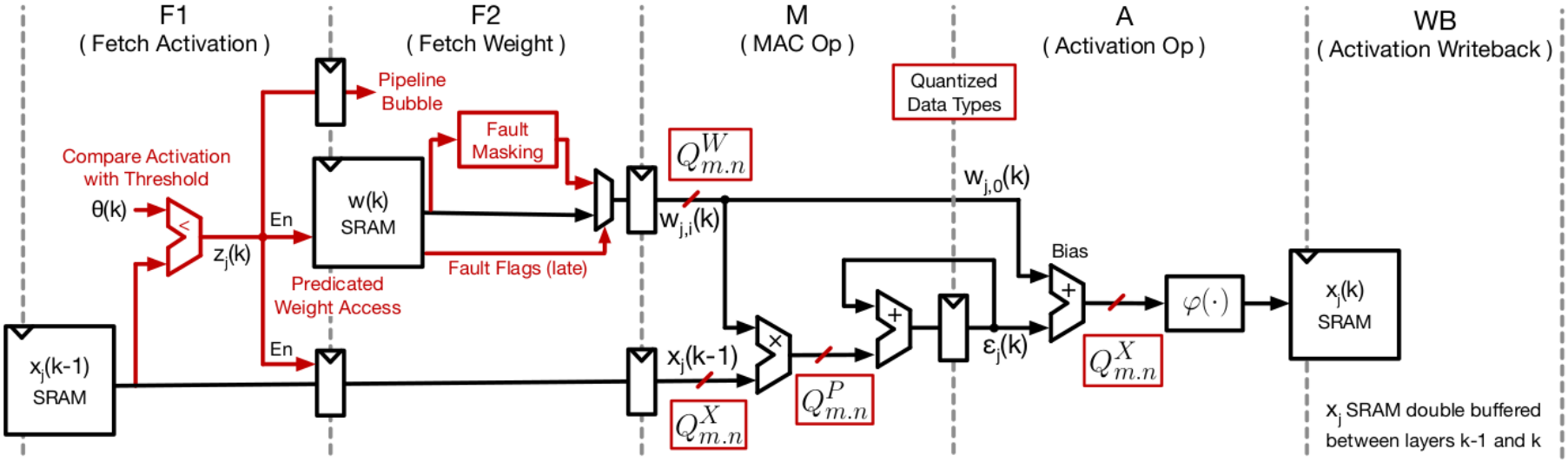


parameters varied

functional unit placement (in in pipeline)

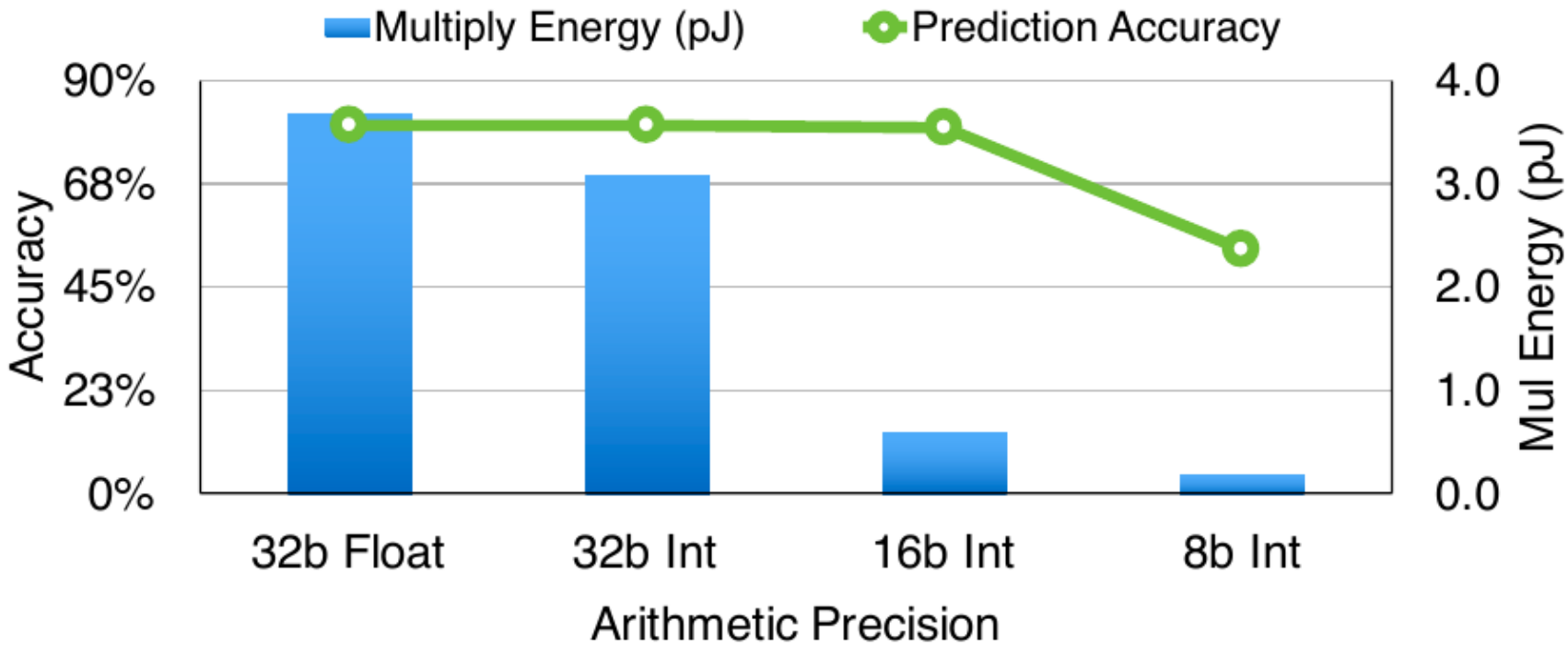
number of lanes

HW pipeline


F1
(Fetch Activation)
F2
(Fetch Weight)
M
(MAC Op)
A
(Activation Op)
WB
(Activation Writeback)
Quantized
Data Types
Pipeline
Bubble
Fault
Masking
Compare Activation
with Threshold
θ(k)
z_j(k)
En
w(k)
SRAM
Predicated
Weight Access
Fault Flags (late)
En
x_j(k-1)
SRAM
Q^W_m.n
w_j,i(k)
w_j,0(k)
Bias
x_j(k-1)
Q^X_m.n
Q^P_m.n
ε_j(k)
Q^X_m.n
φ(·)
x_j(k)
SRAM
x_j SRAM double buffered
between layers k-1 and k

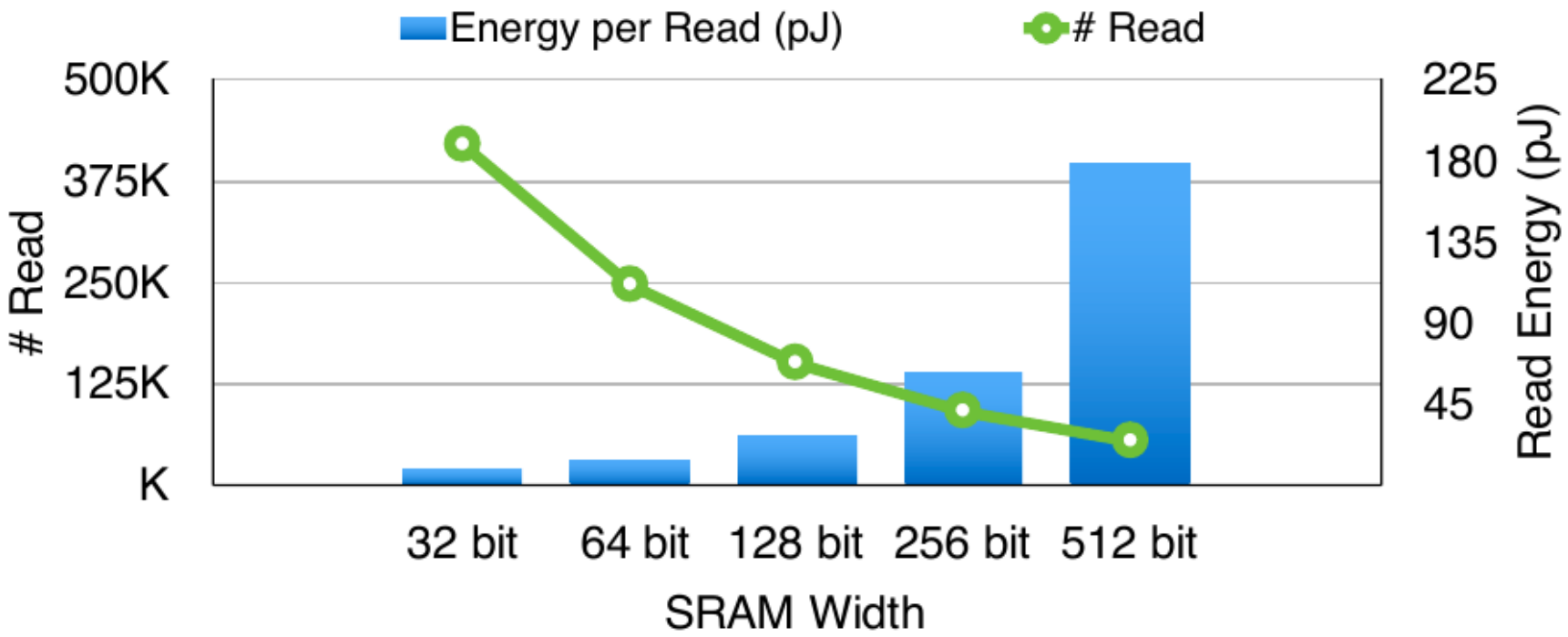
Decreasing precision (1)

from another neural network ASIC accelerator paper:



Decreasing precision (2)

from another neural network ASIC accelerator paper:



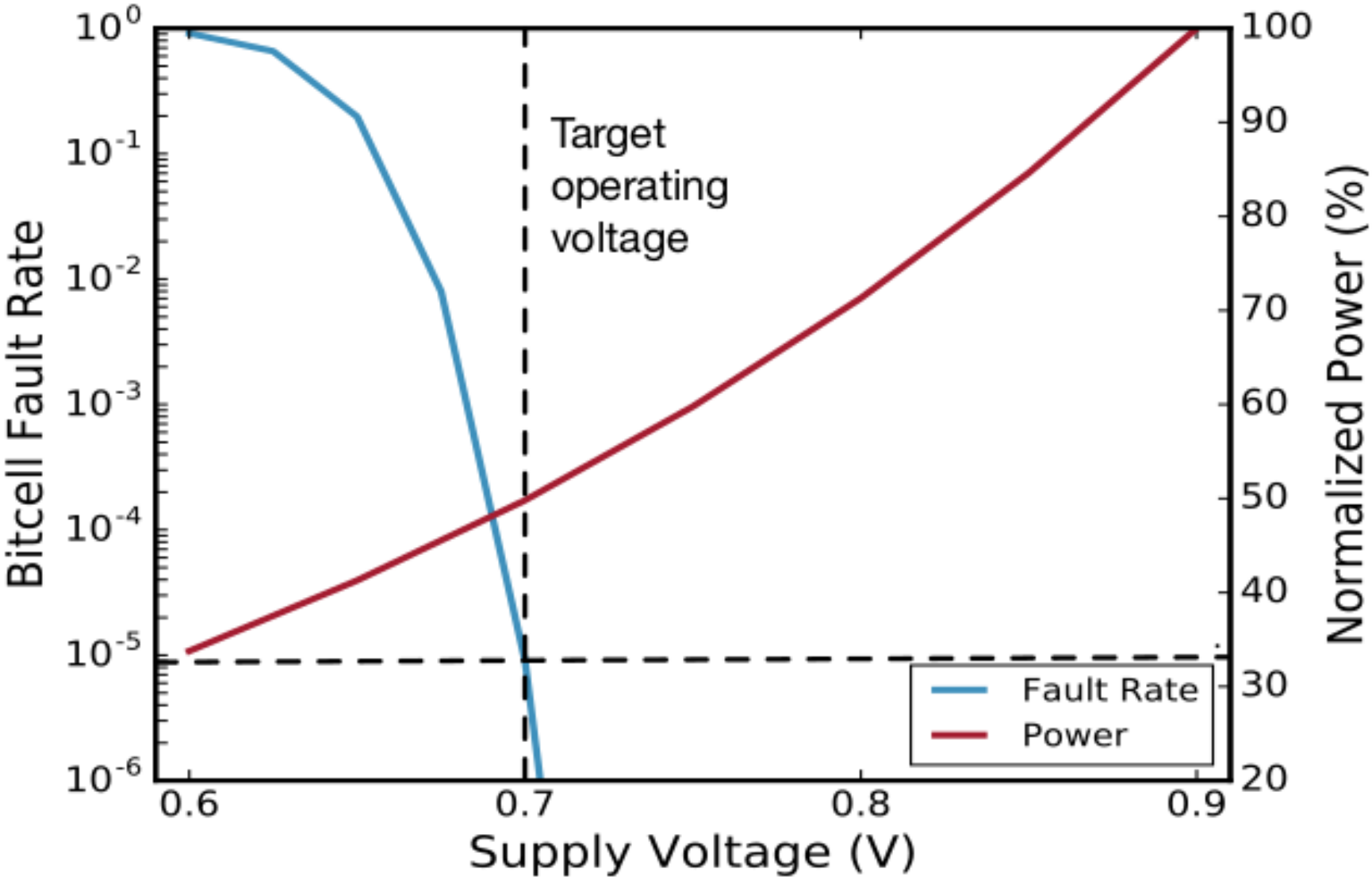
Pruning

short-circuit calculations close to zero

statically — remove neurons with almost all zero weights

dynamically – compute 0 if input is near-zero without checking weights

SRAM danger zone



Traditional reliability techniques

don't run at low voltage/etc.

redundancy — error correcting codes

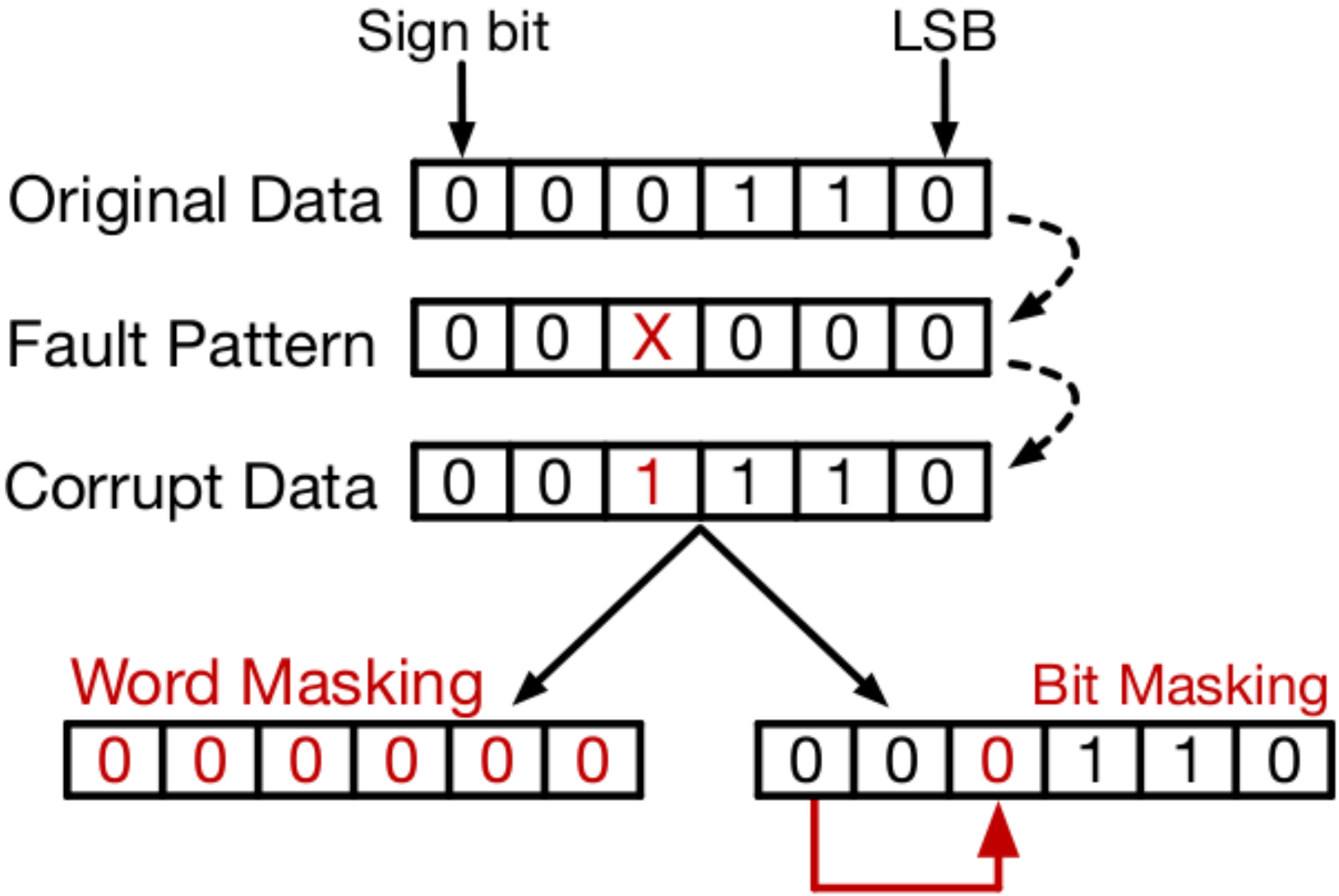
Algorithmic fault handling

calculations are approximate anyways

“noise” from imprecise training data, rounding, etc.

physical faults can just be more noise

round-down on faults



design exploration

huge number of variations:

amount of parallel computations

width of computations/storage

size of models

best power per accuracy

note: other papers on this topic

EIE — same conference

omitted zero weights in more compact way

noted: lots of tricky branching on GPUs/CPUs.

solved general sparse matrix-vector multiply problem

design tradeoffs in the huge

next time: Warehouse-Scale Computers

AKA datacenters — most common modern supercomputer

no paper review

reading on schedule: Barroso et al, The Datacenter as a Computer, chapters 1 and 3 and 6

next week — security

general areas of HW security:

protect programs from each other — page tables, kernel mode, etc.

protect programs from adversaries — bounds checking, etc.

protect programs from people manipulating the hardware

next week's paper: last category