CMPE110 Lecture 25

Custom Hardware Designs

Heiner Litz

https://canvas.ucsc.edu/courses/19290

Announcements

- No Office hours today! (Grad Open House)
- Instead: Office hours on Monday 2:30-3:30
- Student Evaluations
  - Important for the quality of this course
  - Submissions are guaranteed to be **confidential**!
  - Important for me
  - Don’t forget to evaluate the TAs

Many techniques to improve Performance

- Deep Pipelining
- Superscalar Execution
- Branch Prediction
- Out-of-order Execution
- Multicore

- What is the problem?
  - Performance comes at a cost
  - All these techniques consume a lot of power

Efficiency

We care about multiple metrics

- Performance
  - ExecutionTime = IC * CPI * CCT, operations/second
- Power = C * Vdd^2 * F
- Energy = C * Vdd^2
- Cost
  - Area, design and testing complexity

Can we improve performance **without** increasing power, energy, and cost?

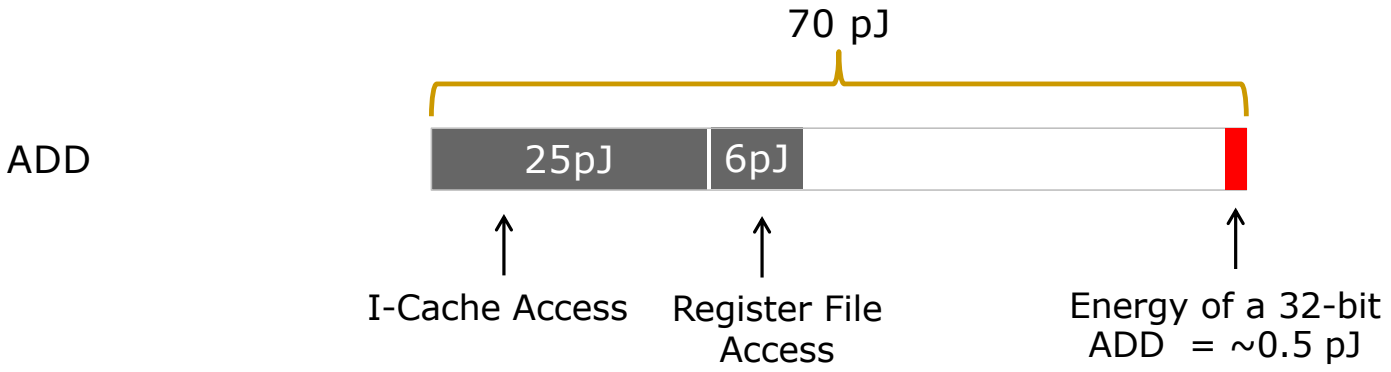
Reminder: Improving Performance without Increasing Power

- So, how do we go about this?
  - Remember: $\textbf{Power} = \textbf{C} * \textbf{Vdd}^2 * \textbf{F}_{0\rightarrow1} + \textbf{Vdd} * \textbf{I}_{\textbf{leakage}}$
- What should we optimize processors for?
  - Answer: energy per instruction (EPI)

$$Power = \frac{energy}{second} = \frac{energy}{instruction} \times \frac{instructions}{second}$$

  - After minimizing EPI, tune performance & power as needed
    - Higher for server, lower for cellphone

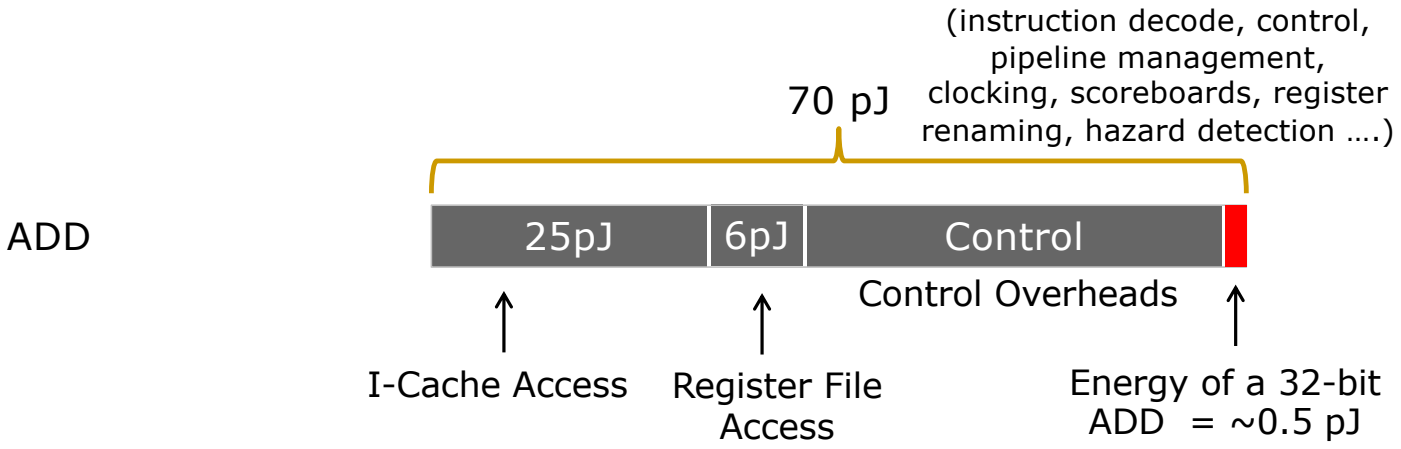
Anatomy of a RISC Instruction



0.7% energy goes to math!

* Assuming a typical 32-bit embedded RISC in 45nm @ 0.9V technology

Anatomy of a RISC Instruction



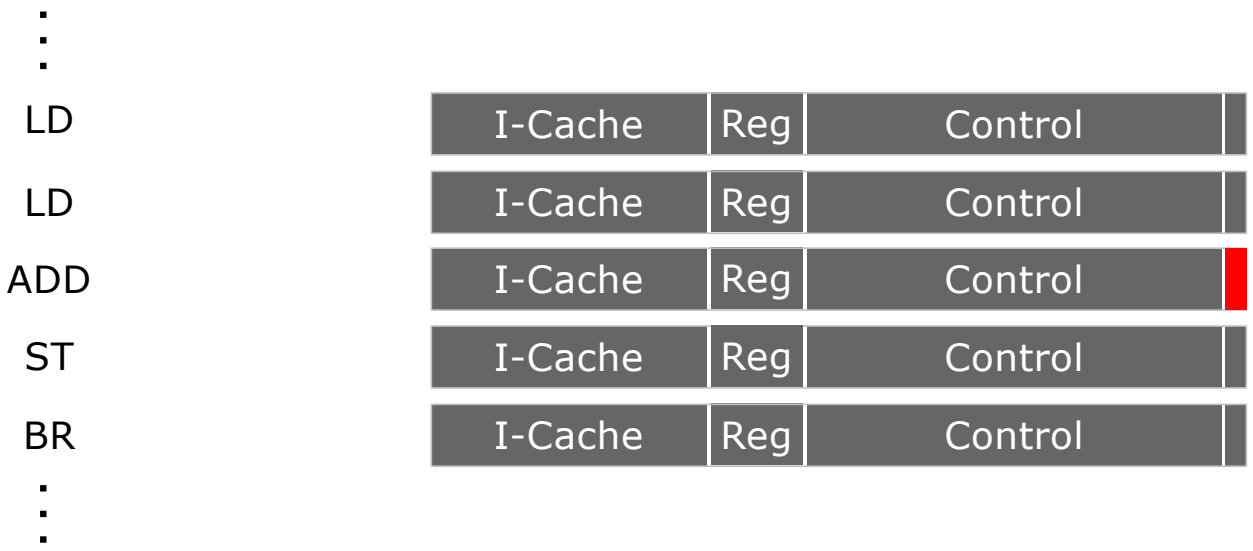
* Assuming a typical 32-bit embedded RISC in 45nm @ 0.9V technology

That's Not All!

ADD

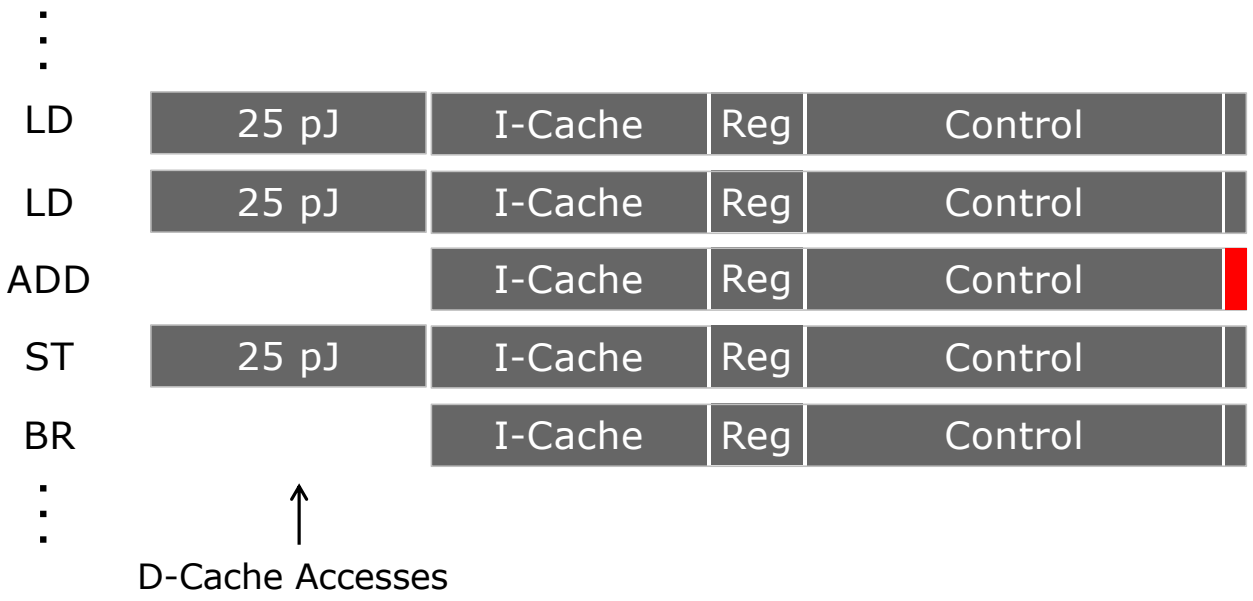
* Assuming a typical 32-bit embedded RISC in 45nm @ 0.9V technology

Don't Forget the Overhead Instructions



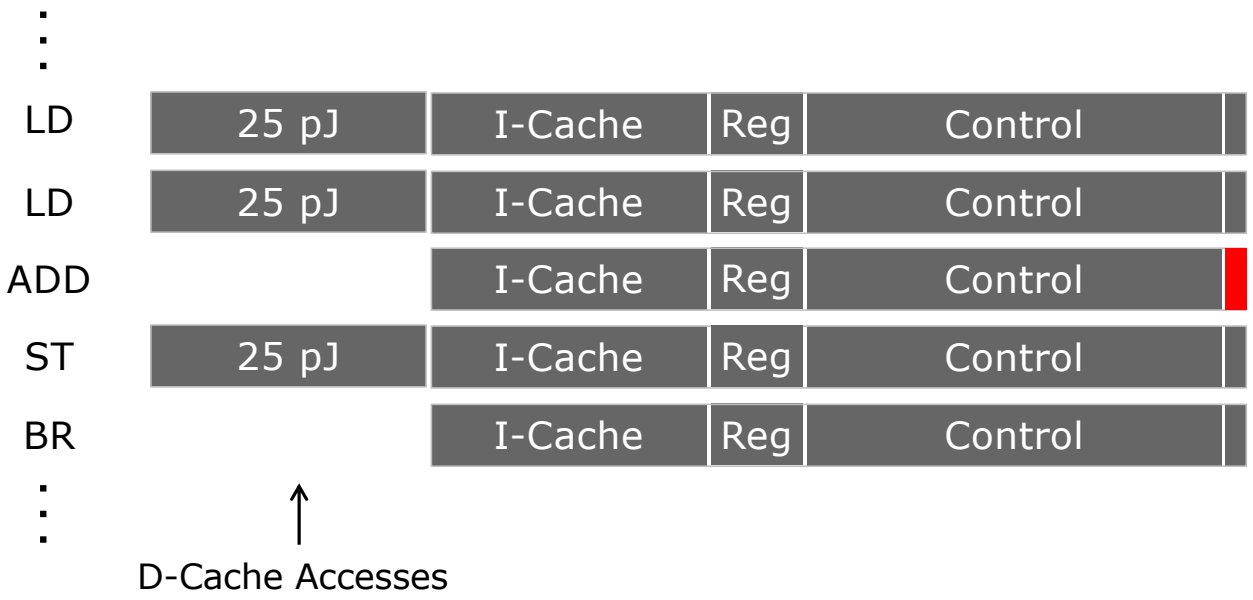
* Assuming a typical 32-bit embedded RISC in 45nm @ 0.9V technology

Don't Forget the Overhead Instructions



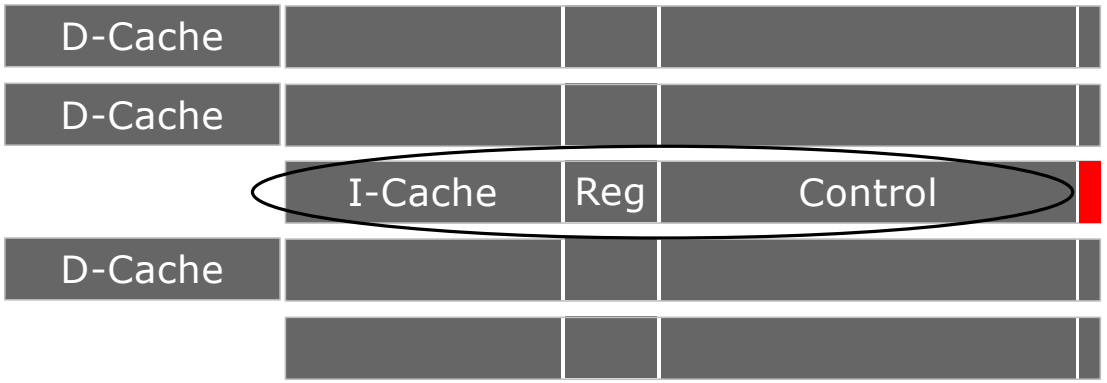
* Assuming a typical 32-bit embedded RISC in 45nm @ 0.9V technology

How Do We Reduce This Waste?



* Assuming a typical 32-bit embedded RISC in 45nm @ 0.9V technology

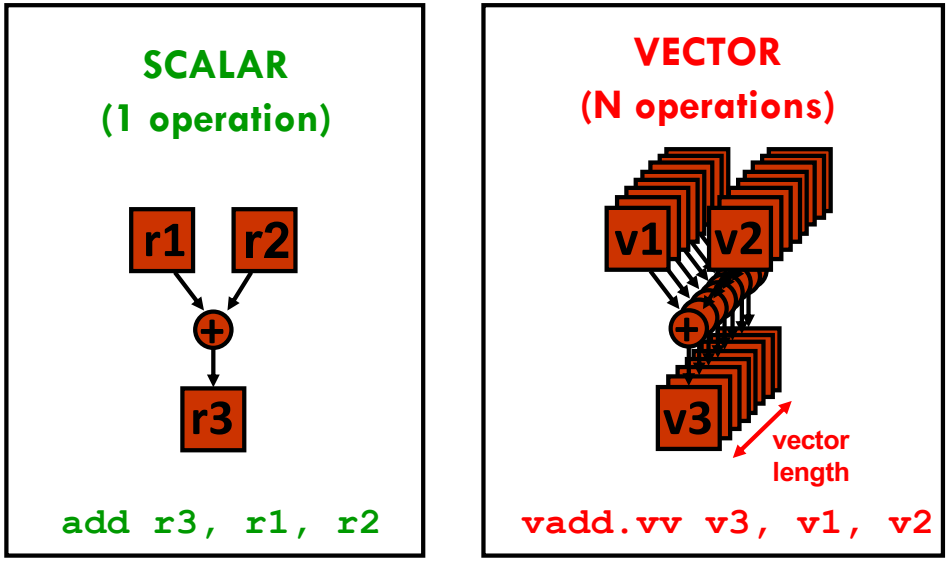
Reducing This Waste



1. Perform a large number of operations per instruction

Have We Done this Already?

Data-level Parallelism (Vectors)

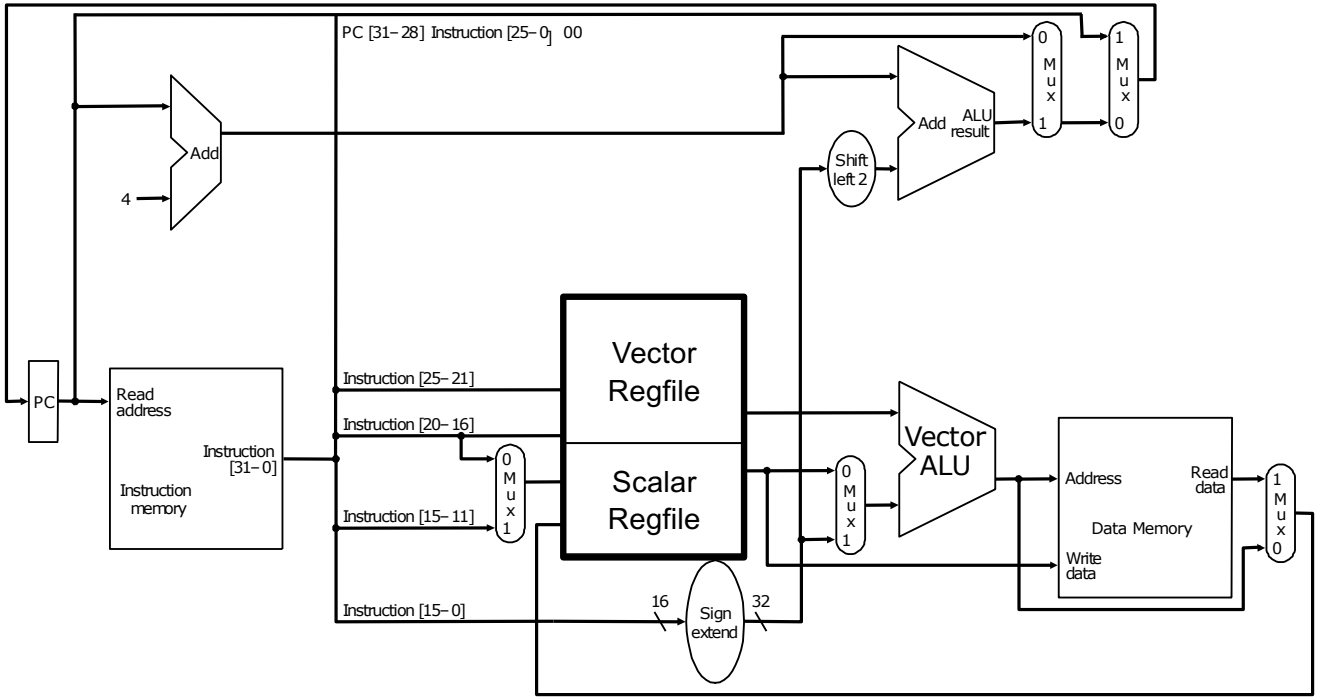


- Scalar instructions operate on single numbers (scalars)
- Vector instructions operate on vectors of numbers
  - Linear sequences of numbers
  - Stored in a vector register file

Benefits of Vector Instructions

- Each instruction specifies more parallel work
  - Can build a higher-performance processor
- Instruction overheads amortized better
  - Overhead: energy for fetch & decode instruction
  - Amortized over multiple arithmetic ops
- Fewer overhead instructions
  - Vector loops

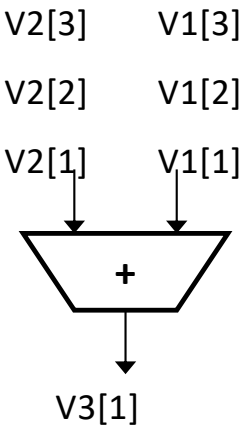
Simple Vector Processor



- Add vector register file
- How do we vectorize the ALU and the control?

Vector ALUs

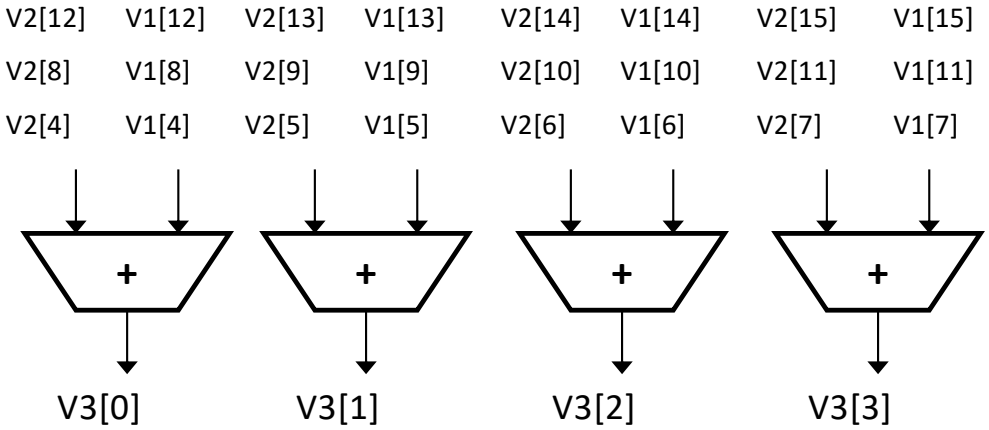
vadd v2,v1 (VL=N)



1 element/cycle

N cycles

1 adder



4 elements/cycle

N/4 cycles

4 adders

Vector/SIMD Summary

- Vector/SIMD can offer around 10x benefit for some apps
- Does SIMD help all applications?

Let’s go back to basics

Programmability vs. Efficiency

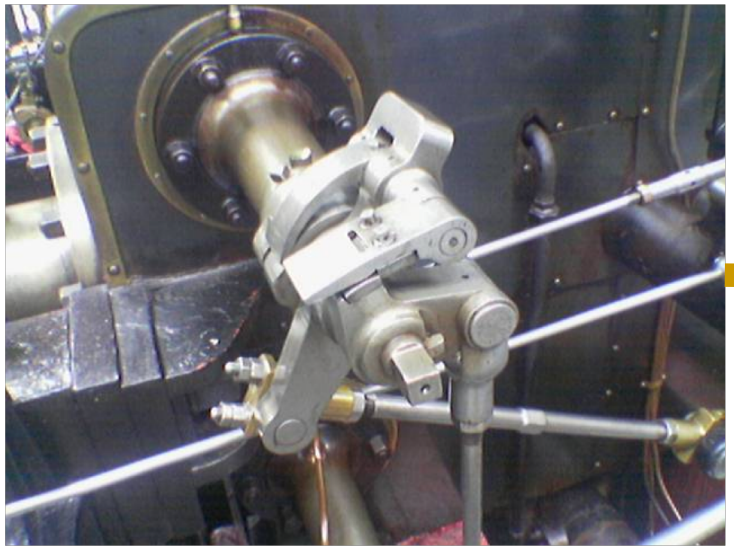
- Processors are highly programmable
  - Can implement any algorithm you can program
  - Cost: Instruction overhead
- Custom hardware is highly efficient
  - Only build the hardware you need
  - Can only compute one algorithm

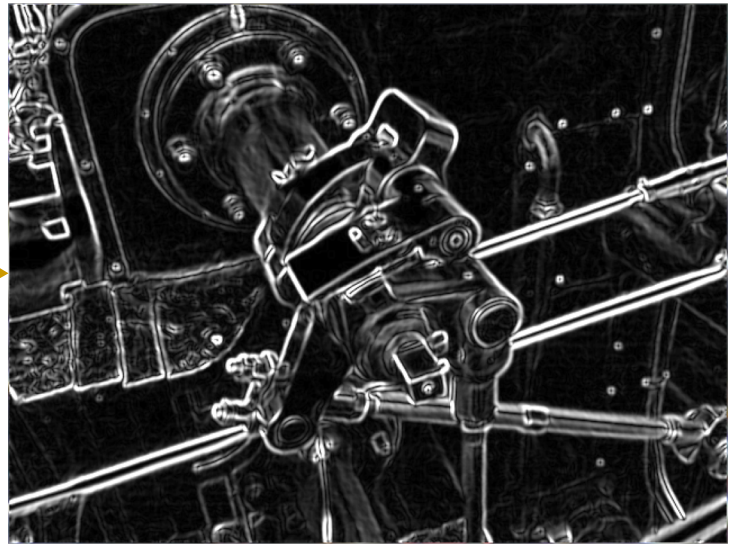
Custom Hardware

- Designed to run one algorithm really fast
- Contains algorithm or domain specific hardware
  - Specific mix of math and control units
  - Specific precision at each stage
  - Stages are cascaded
    - 1 instruction, 1 writeback

Sobel Filter

- Image processing and computer vision
- Performs edge detection
- 2D-Convolution with a filter



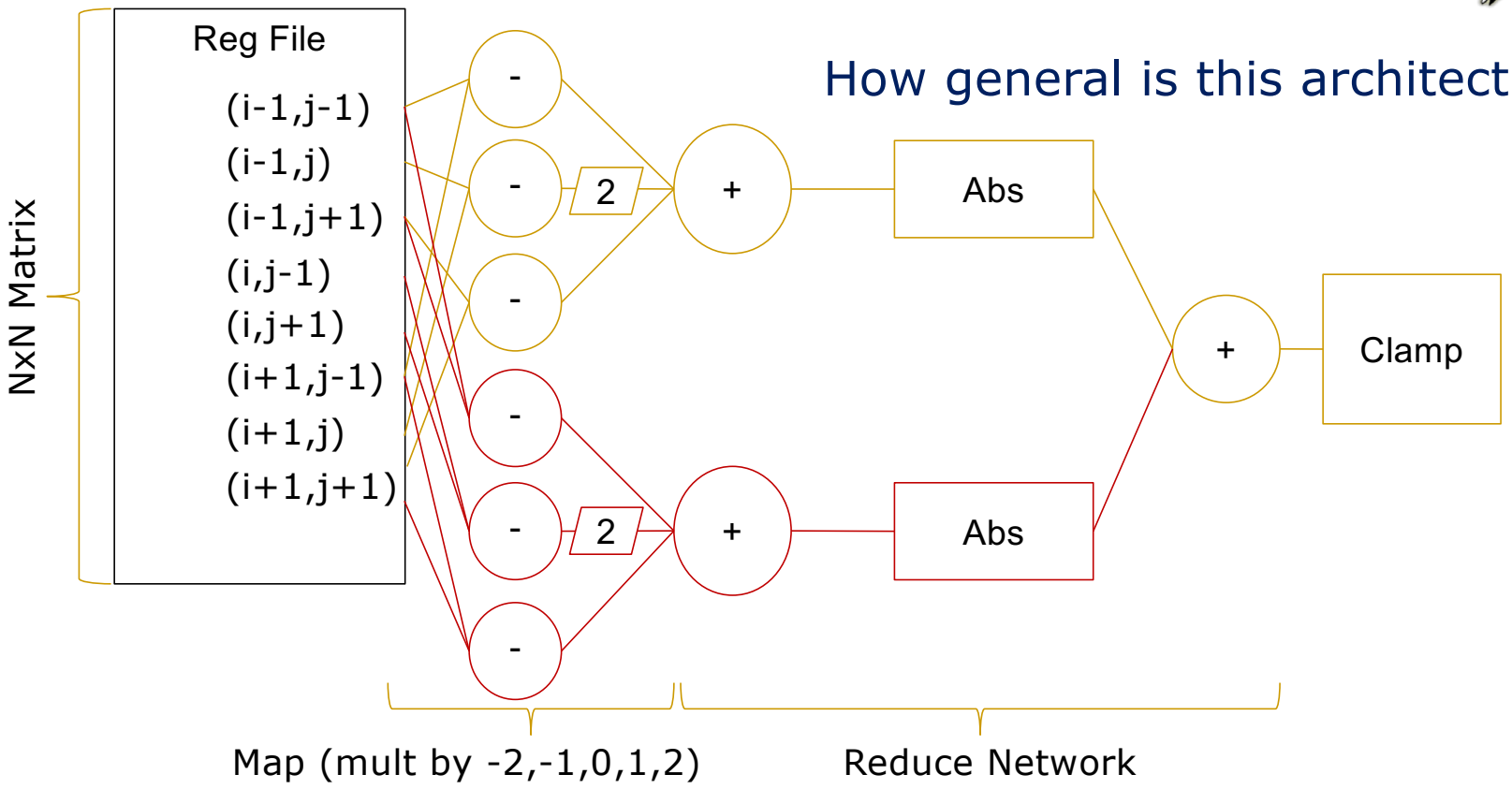


Sobel Filter

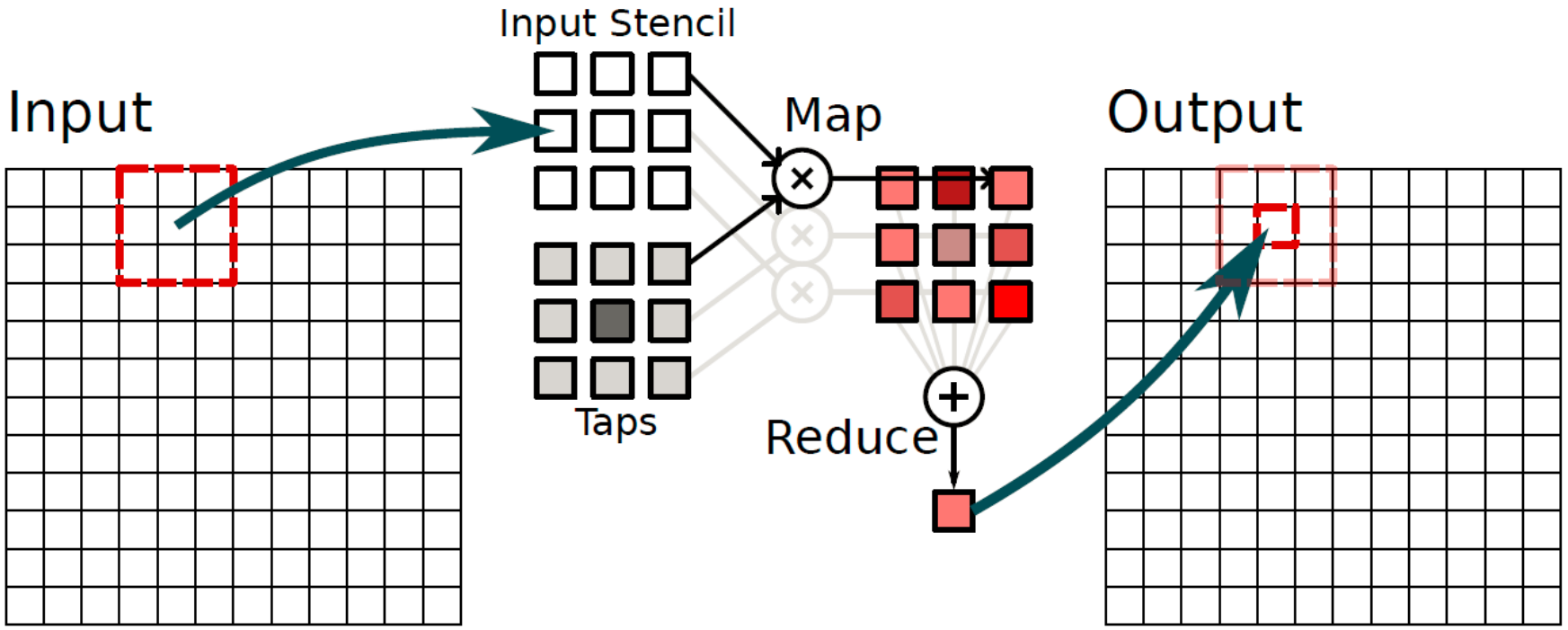
- Arithmetic operations
  - 4 shifts
  - 10 adds/sub (accumulate)
  - 2 abs
  - 1 value clamp (comparison)
- How many instructions?

```
//Calculate the x convolution
 for (int i=1; i<gray_img.rows; i++) {
   for (int j=1; j<gray_img.cols; j++) {
     sobel = abs(gray_img.data[IMG_WIDTH*(i-1) + (j-1)]-
              gray_img.data[IMG_WIDTH*(i+1) + (j-1)] +
              2*gray_img.data[IMG_WIDTH*(i-1) + (j)] -
              2*gray_img.data[IMG_WIDTH*(i+1) + (j)] +
              gray_img.data[IMG_WIDTH*(i-1) + (j+1)] -
              gray_img.data[IMG_WIDTH*(i+1) + (j+1)]);

    sobel += abs(gray_img.data[IMG_WIDTH*(i-1) + (j-1)] -
              gray_img.data[IMG_WIDTH*(i-1) + (j+1)] +
              2*gray_img.data[IMG_WIDTH*(i) + (j-1)] -
              2*gray_img.data[IMG_WIDTH*(i) + (j+1)] +
              gray_img.data[IMG_WIDTH*(i+1) + (j-1)] -
              gray_img.data[IMG_WIDTH*(i+1) + (j+1)]);

    sobel = (sobel > 255) ? 255 : sobel;

img_out.data[IMG_WIDTH*(i) + j] = sobel;
   }
 }
```

Domain Specific HW



Convolution Engine



Domain Specific Hardware

	Map	Reduce	Stencil Size	Data Flow
IME SAD	Abs Diff	Add	4x4	2D Convolution
FMW ½ pixel up-sample	Multiply	Add	6	1D Horizontal & vertical conv.
FME ¼ pixel up-sample	Average	None	--	2D Matrix operation
SIFT Gaussian blur	Multiply	Add	9, 13, 15	1D Horizontal & vertical conv.
SIFT DoG	Subtract	None	--	2D Matrix operation
SIFT Extreme	Compare	Logic AND	3	1D Horizontal & vertical conv.
Demosaic interpolation	Multiply	Complex	3	1D Horizontal & vertical conv.

- Find similar computations within a domain
- Sacrifice some efficiency for generality within domain

Memory Concerns: Bandwidth Matching

- Can only operate as fast as we can feed it
  - Memory too slow: Accelerator must stall
  - Accelerator too slow: Add more accelerators (or make yours bigger!)
- Match performance with available memory bandwidth

Memory Overheads

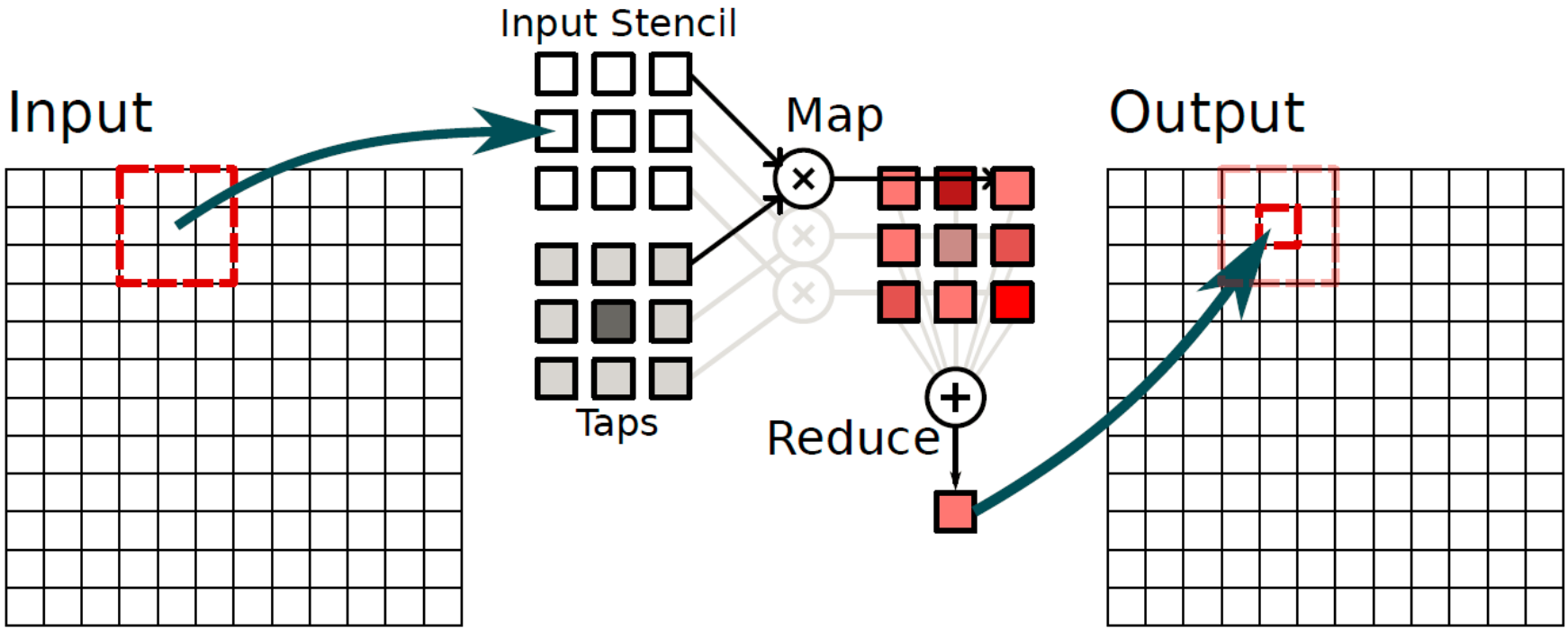
Operation	Energy	Scale
8-bit add	0.03 pJ	1
32-bit add	0.10 pJ	3
32-bit FP mult	4.00 pJ	133
RISC instruction	**70.00 pJ**	**2,300**
8KB cache access	10.00pJ	300
1MB cache access	**100.00 pJ**	**3,000**
DRAM access	**2,000.00 pJ**	**60,000**

Minimizing Memory Access

- Getting data to and from accelerators (and cores) is expensive
  - Especially if that data is in DRAM
- To minimize energy
  - Minimize read/writes by buffering operands and intermediates locally
  - Maximize data reuse (where do we find reuse in Sobel?)

Read and write once, reuse often!

Convolution Engine



- How many rows of data do we need to buffer?
- How much new data do we need per pixel?

Outlook

- End of Moore's Law is in sight
- All architectures are now power limited
- Multi-core is running out of steam
- Heterogenous Custom Hardware Accelerators to the Rescue
  - Energy efficient and high performance
  - Difficult to design (needs new HW for each application domain)
  - Hard to program
- Got interested?
  - Advanced logic design (CE125)
  - Advanced Computer Architecture (202), Accelerators (293), Parallel programming (220)