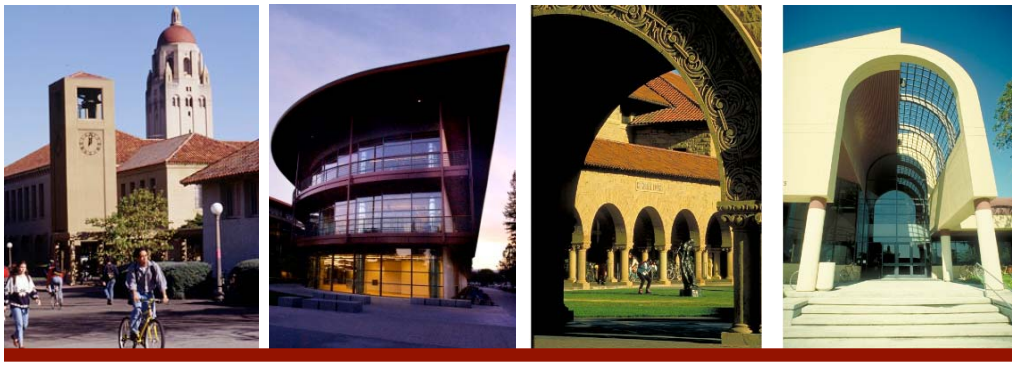
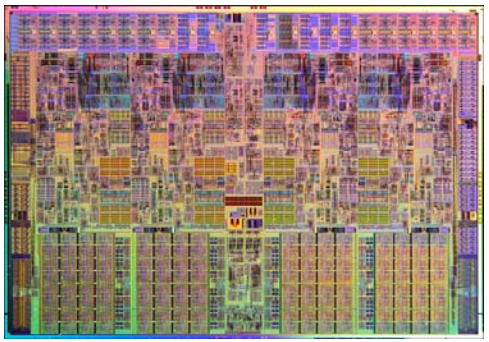


Understanding Sources of Inefficiency in General-Purpose Chips

Rehan Hameed
Wajahat Qadeer
Megan Wachs
Omid Azizi
Alex Solomatnikov
Benjamin Lee
Stephen Richardson
Christos Kozyrakis
Mark Horowitz

GP Processors Are Inefficient



Nehalem

vs.

Emerging Applications

Processors work well for a broad range of applications

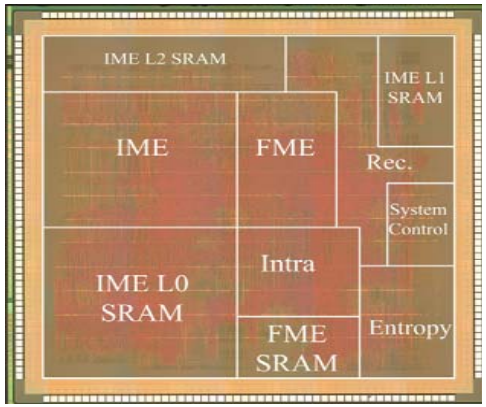
- Have well amortized NRE
- For a specific performance target, energy and area efficiency is low

Processors are power limited

- Hard to meet performance and energy of emerging applications
  - Enhancement of low-quality video, analysis and capture motion in 3D, etc
- At fixed power, more ops/sec requires lower energy/op

More Efficient Computing Is Possible

Emerging Applications



ASIC

Embedded media devices perform GOP/s

- Cell phones, video cameras, etc

Efficiency of processors inadequate for these devices

- ASICs needed to meet stringent efficiency requirements

ASICs are difficult to design and inflexible

An Example

High definition video encoding is ubiquitous

- Cell phones, camcorders, point and shoot cameras, etc.

A small ASIC does it

- Can easily satisfy performance and efficiency requirements

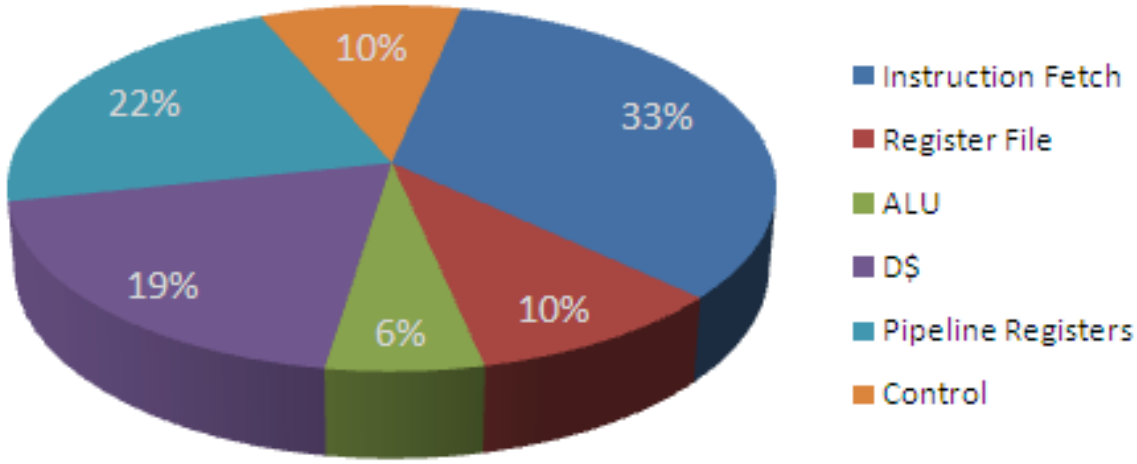
Very challenging for processors

- What makes the processors inefficient compared to ASICs?
- What does it to take to make a processor as efficient as an ASIC?
- How much programmability do you lose?

CMP Energy Breakdown

For HD H.264 encoder

- 2.8GHz Pentium 4 is 500x worse in energy*
- Four processor Tensilica based CMP is also 500x worse in energy*



Assume everything but functional unit is overhead

- Only 20x improvement in efficiency

* Chen, T.-C., et al., "Analysis and architecture design of an HDTV720p 30 frames/s H.264/AVC encoder," Circuits and Systems for Video Technology, IEEE Transactions on, vol.16, no.6, pp. 673-688, June 2006.

Achieving ASIC Efficiencies: Getting to 500x

Need basic ops that are extremely low-energy

- Function units have overheads over raw operations
- 8-16 bit operations have energy of sub pJ
  - Function unit energy for RISC was around 5pJ

And then don't mess it up

- "No" communication energy / op
  - This includes register and memory fetch
- Merging of many simple operations into mega ops
  - Eliminate the need to store / communicate intermediate results

How Much Specialization Is Needed?

How far will general purpose optimizations go?

- Can we stay clear of application specific optimizations?
- How close to ASIC efficiencies will this achieve?

Better understand nature of various overheads

- What are the “long poles” that need to be removed

Is there an incremental path from GP to ASIC

- Is it possible to create an intermediate solution?

Case Study

Use Tensilica to create optimized processors

Transform CMP into an efficient HD H.264 encoder

- To better understand the sources of overhead in processor

Why H.264 Encoder?

- It's everywhere
- Variety of computation motifs – *data parallel to control intensive*
- Good software and hardware implementations exist
  - ASIC H.264 solutions demonstrate a large energy advantage

Optimization Strategy For Case Study

Two optimization stages

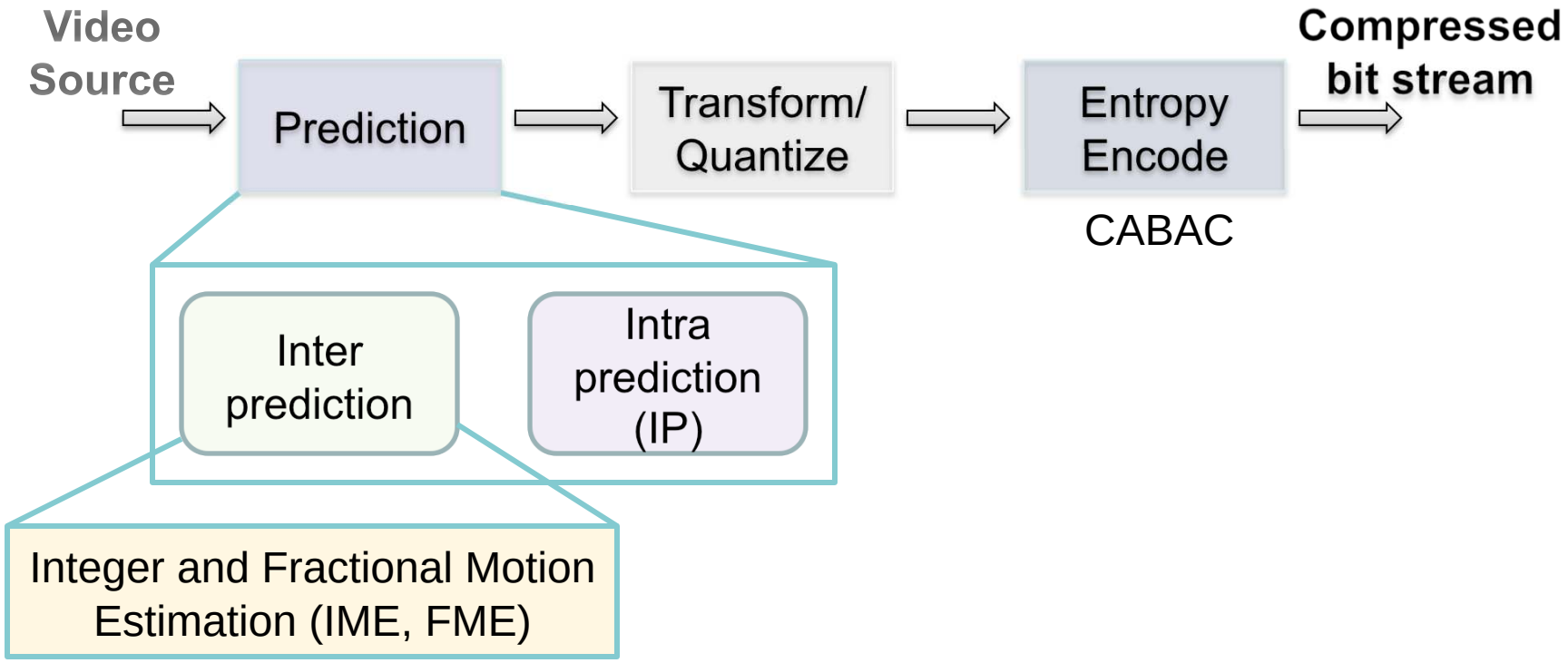
- General purpose, data parallel optimizations
  - SIMD, VLIW, reduced register and data path widths
  - Operation fusion – *limited to two inputs and one output*
    - Similar to Intel's SSE instructions

- Application specific optimizations
  - Arbitrary new compute operations
  - Closely couple data storage and data-path structures

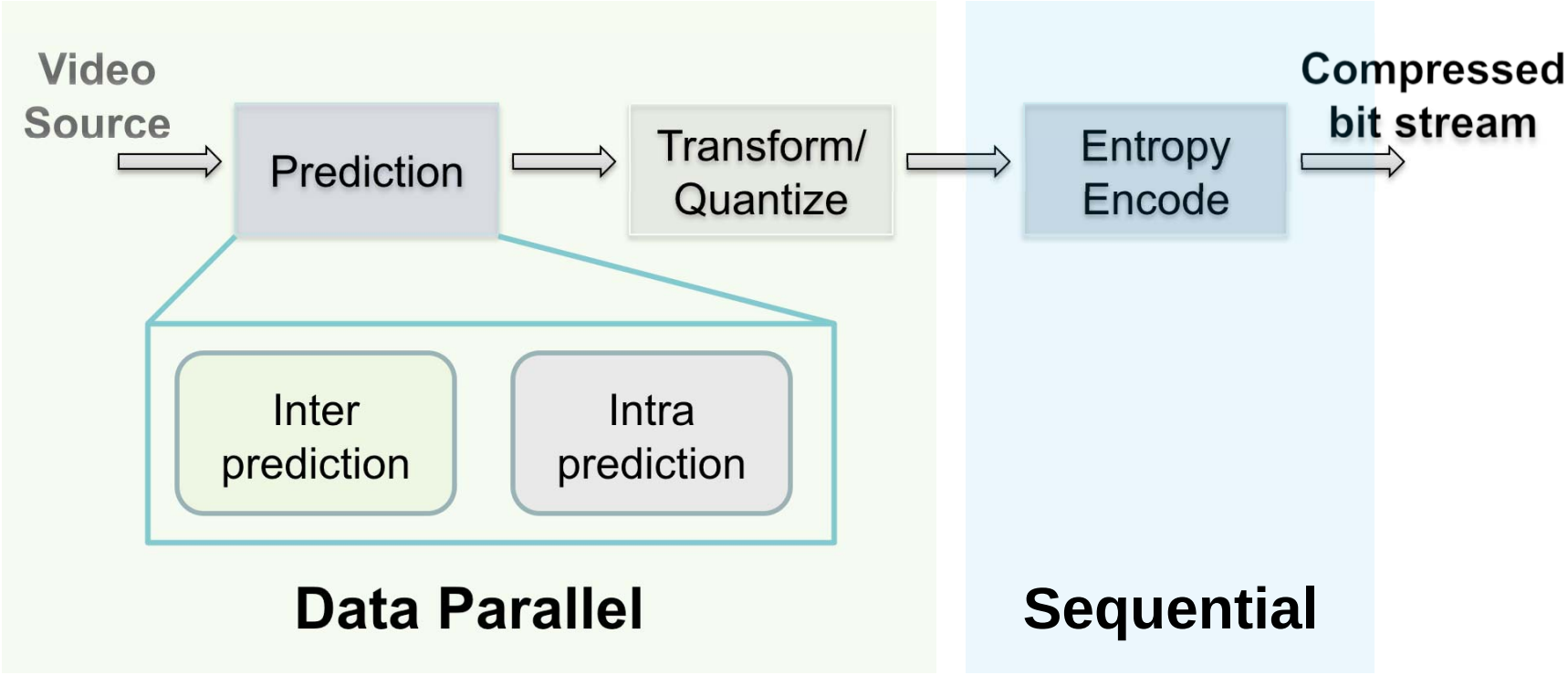
What Is H.264?

Industry standard for video compression

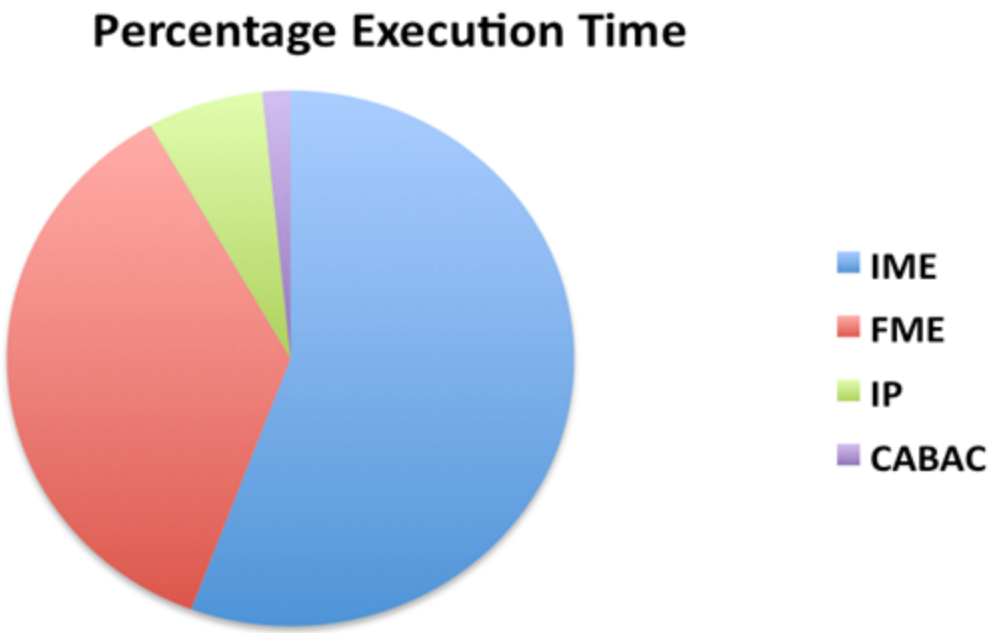
- Digital television, DVD-video, mobile TV, internet video, etc.



Computational Motifs Mapping



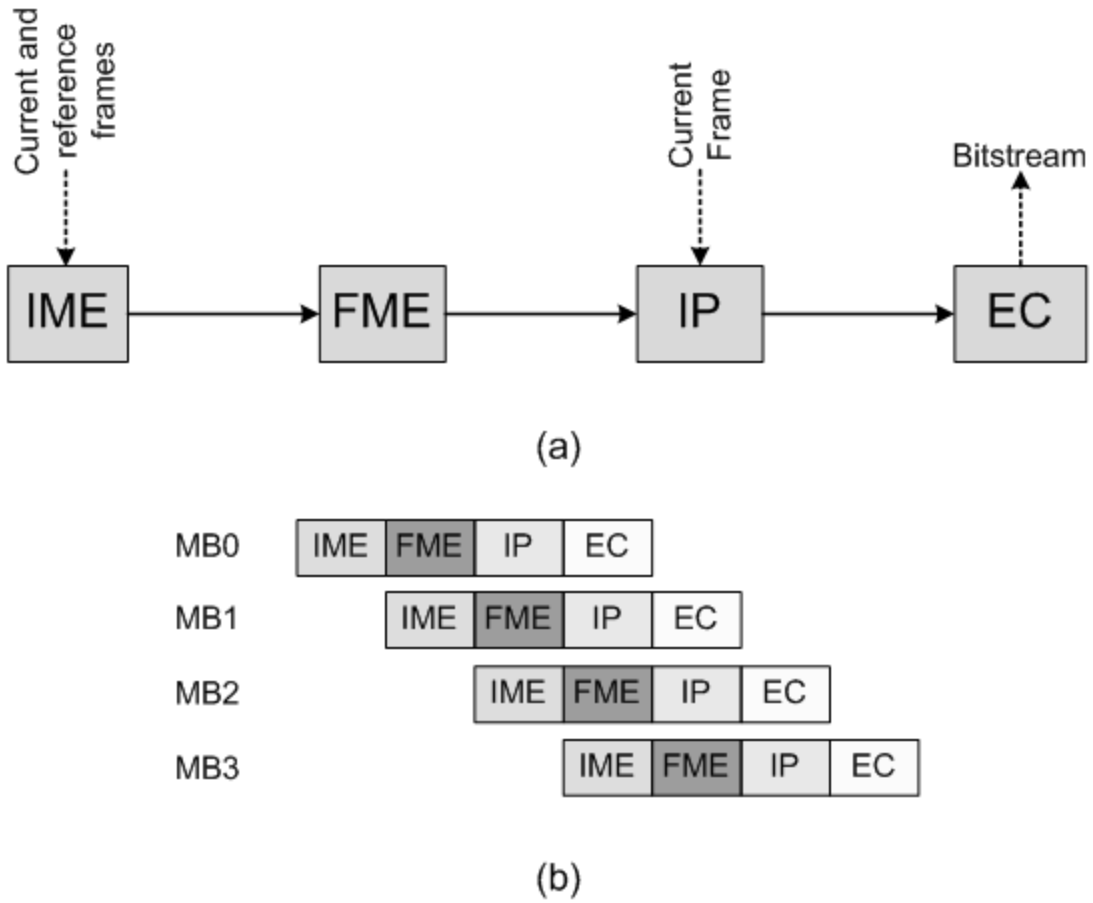
H.264 Encoder - Uni-processor Performance



IME and FME dominate total execution time

CABAC is small but dictates final gain

H.264 – Macroblock Pipeline



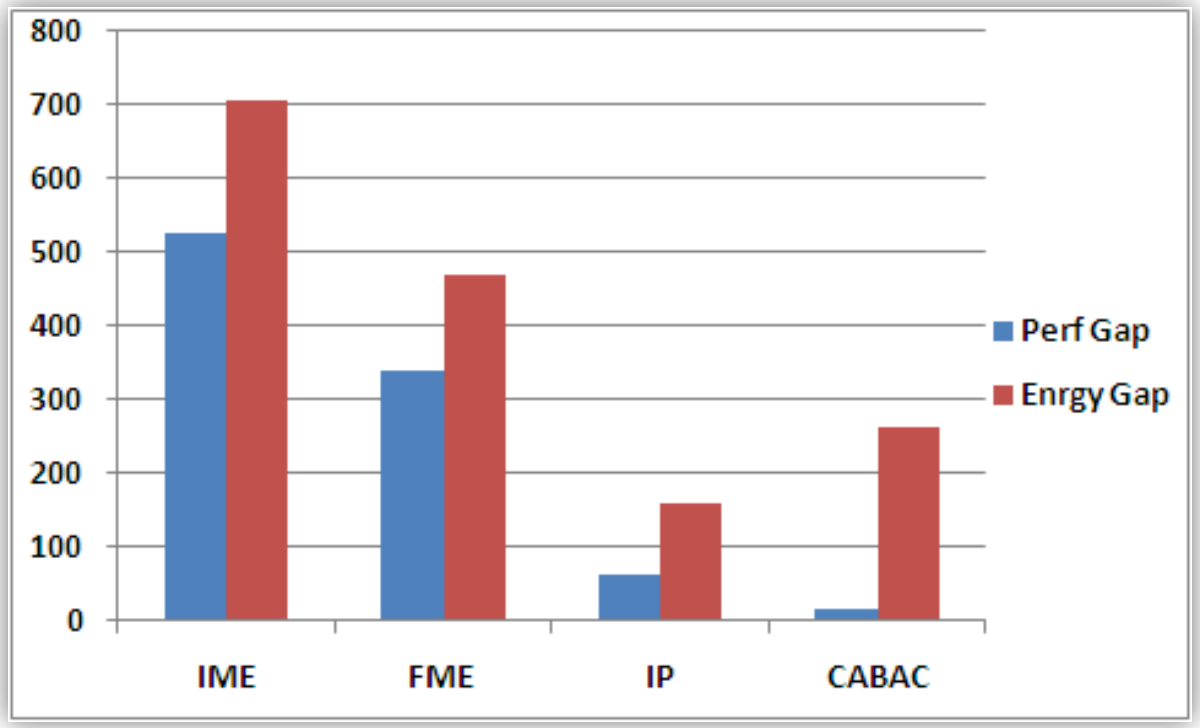
Base CMP vs. ASIC

Huge efficiency gap

- 4-proc CMP 250x slower
- 500x extra energy

Manycore doesn't help

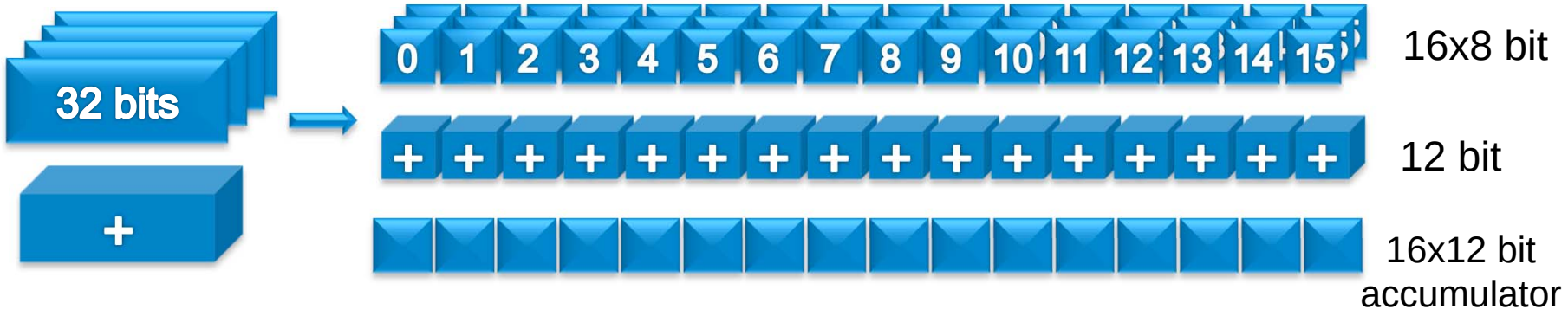
- Energy/frame remains same
- Performance improves



General Purpose Extensions: SIMD & ILP

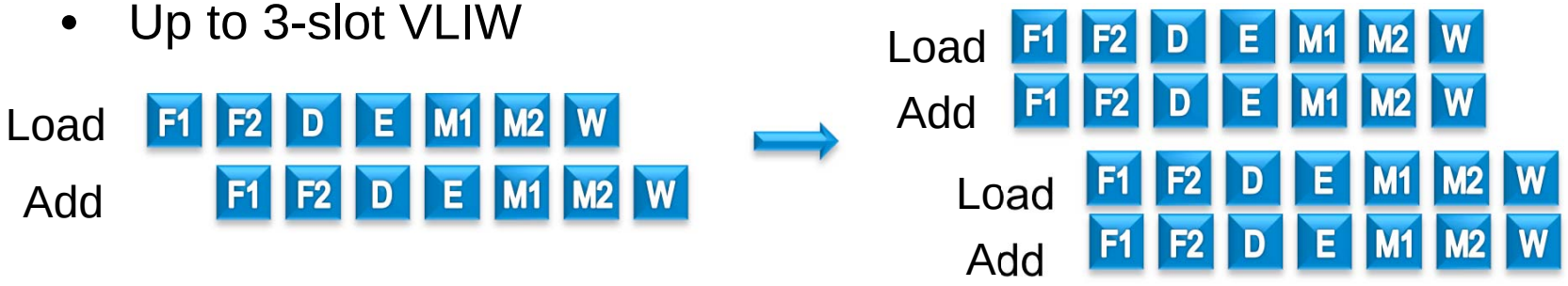
SIMD

- Up to 18-way SIMD in reduced precision

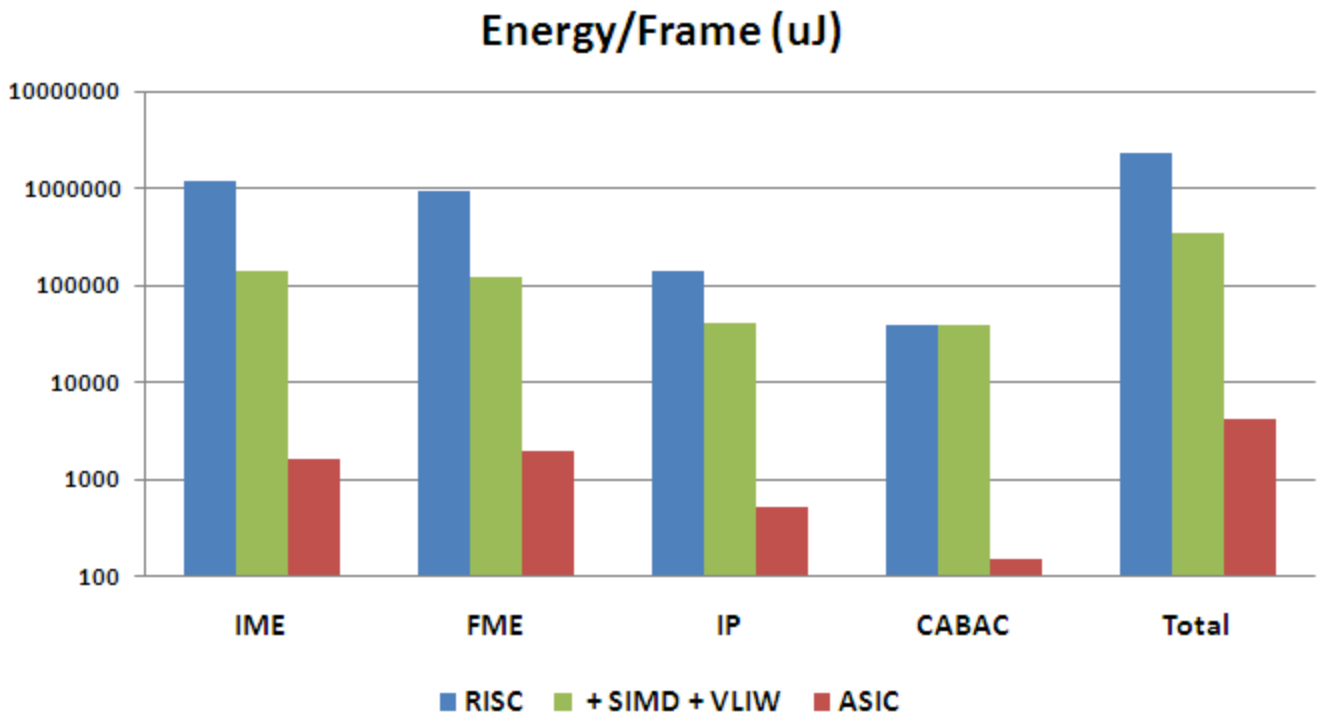


VLIW

- Up to 3-slot VLIW



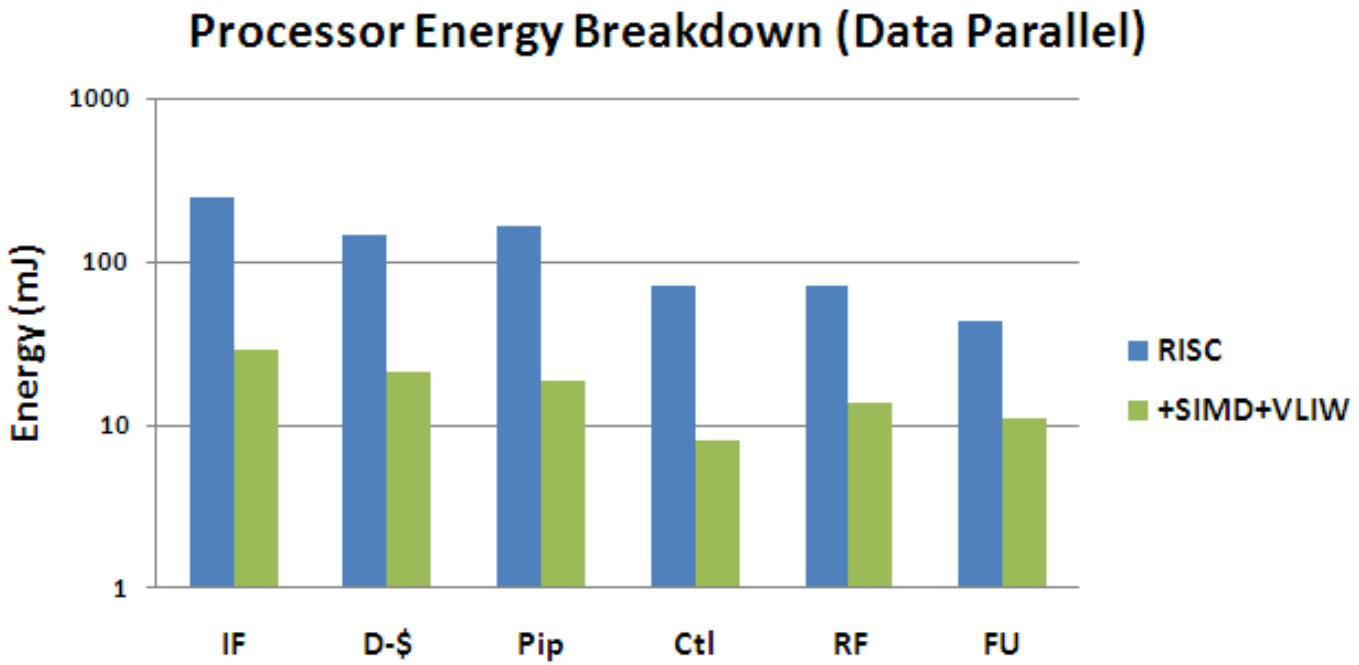
SIMD and ILP - Results



Order of magnitude improvement in performance, energy

- For data parallel algorithms
- But ASIC still better by roughly 2 orders of magnitude

SIMD and ILP – Results



Good news: we made the FU more efficient

- Reduced the power of the op by 4x
  - By bit width / simplification

Bad news: overhead decreased by only 2x

Most of energy dissipation is still an overhead

Operation Fusion

Compiler can find interesting instructions to merge

- Tensilica's Xpres

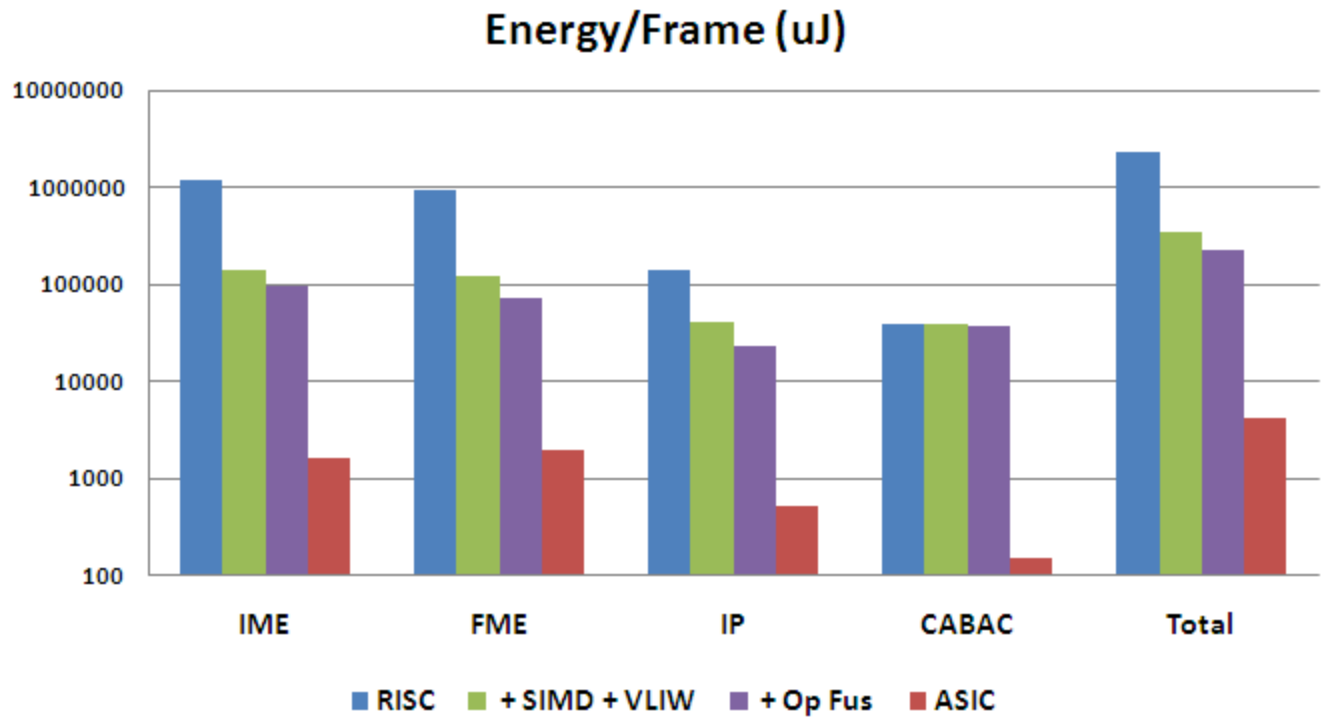
We did this manually

- Tried to create instructions that might be possible

Might be free in future machines

- Common instruction might be present in GP

Operation Fusion – Not A Big Gain



Helps a little, so it is good if free …

50x less energy efficient and 25x slower ASIC

Data Parallel Optimization Summary

Great for data parallel applications

- Improve energy efficiency by 10x over CPU
- But CABAC largely remains unaffected

Overheads still dominate

- Basic operations are very low-energy
- Even with 15-20 operations per instruction, get 90% overhead
- Data movement dominates computation

To get ASIC efficiency need more compute/overhead

- Find functions with large compute/low communication
- Aggregate work in large chunks to create highly optimized FUs
- Merge data-storage and data-path structures

TIME TO START OVER

“Magic” Instructions

Fuse computational unit to storage

Create specialized data storage structures

- Require modest memory bandwidth to keep full
- Internal data motion is hard wired
- Use all the local data for computation

Arbitrary new low-power compute operations

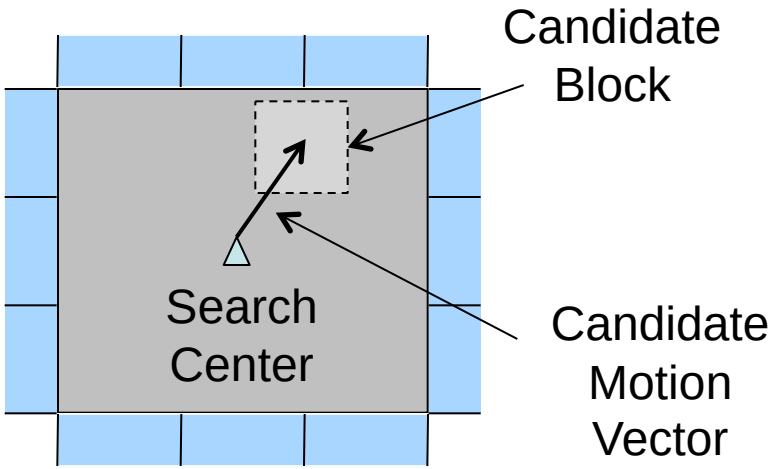
Large effect on energy efficiency and performance

Magic Instructions – *SAD*

$$sum = sum + abs(x_{ref} - x_{cur})$$

Looking for the difference between two images

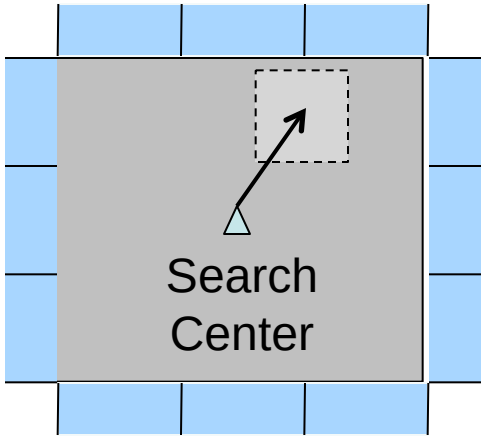
- Hundreds of SAD calculations to get one image difference
  - Need to test many different position to find the best
- Data for each calculation is nearly the same



Magic Instructions - SAD

SIMD implementation

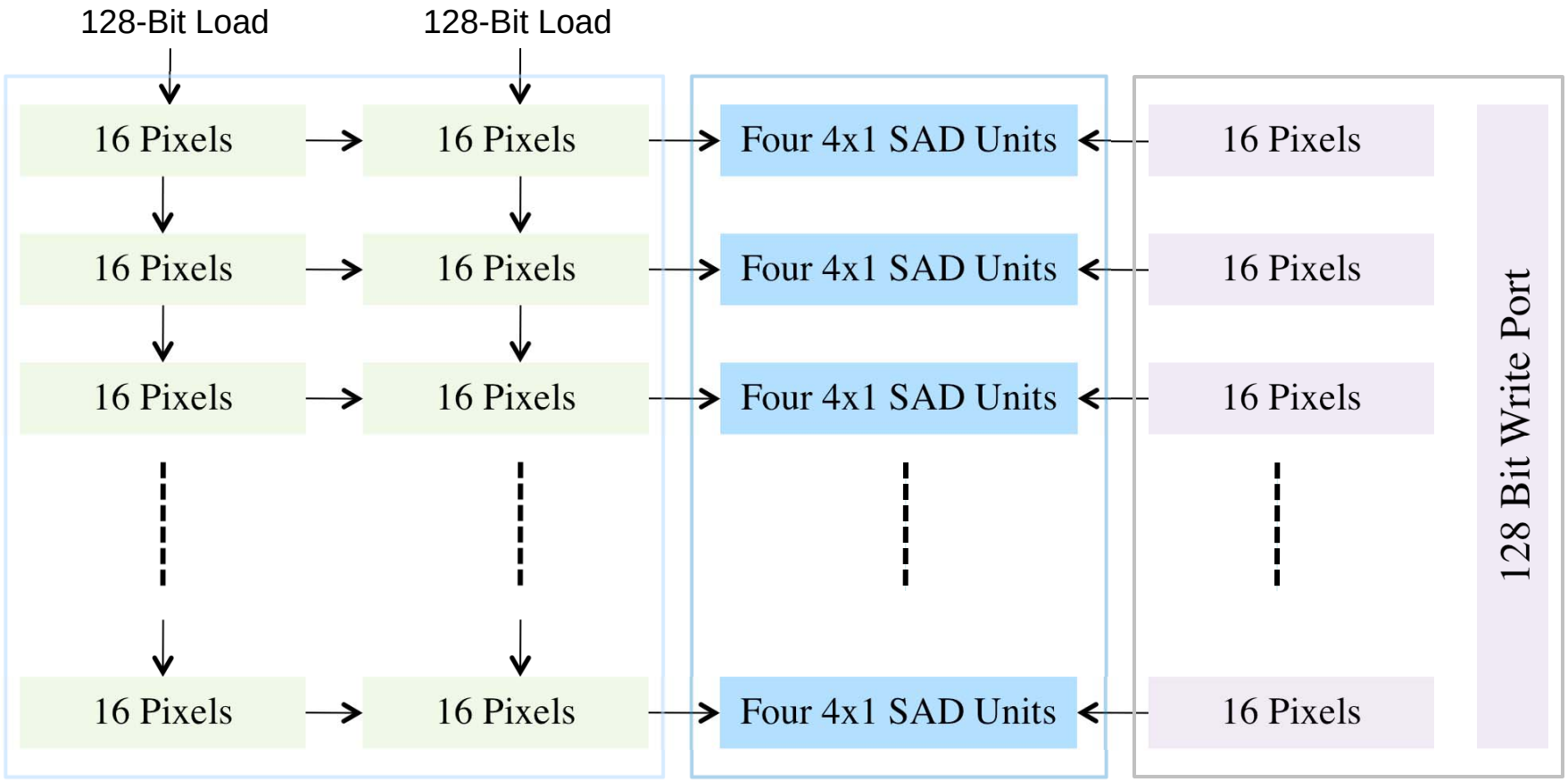
- Limited to 16 operations per cycle
- Horizontal data-reuse requires many shift operations
- No vertical data reuse means wasted cache energy
- Significant register file access energy



Magic – *Serial in, parallel out structure*

- Enables 256 SADs/cycle which reduces fetch energy
- Vertical data-reuse results in reduced DCache energy
- Dedicated paths to compute reduce register access energy

Custom SAD instruction Hardware



Reference Pixel Registers:

Horizontal and vertical shift with parallel access to all rows

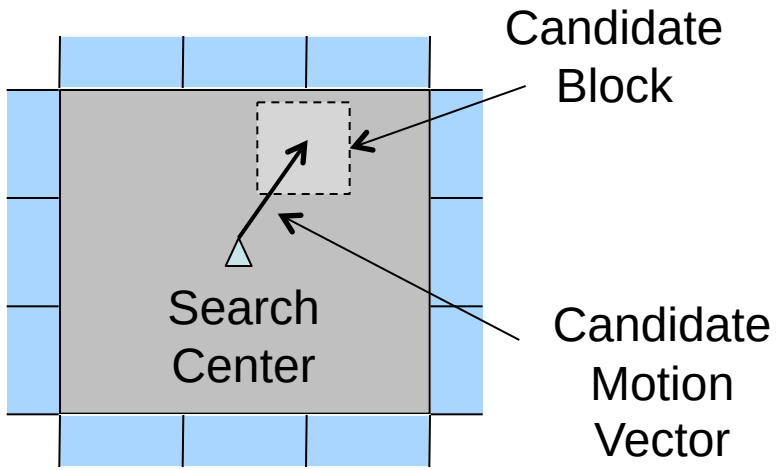
256 SAD Units

Current Pixel Registers

Fractional Motion Estimation

Take the output from the integer motion estimation

- Run again against base image shifted by ¼ of a pixel
- Need to do this in X and Y



Generating the Shifted Images: *Pixel Upsampling*

$$x_n = x_{-2} - 5x_{-1} + 20x_0 + 20x_1 - 5x_2 + x_3$$

FIR filter requiring one new pixel per computation

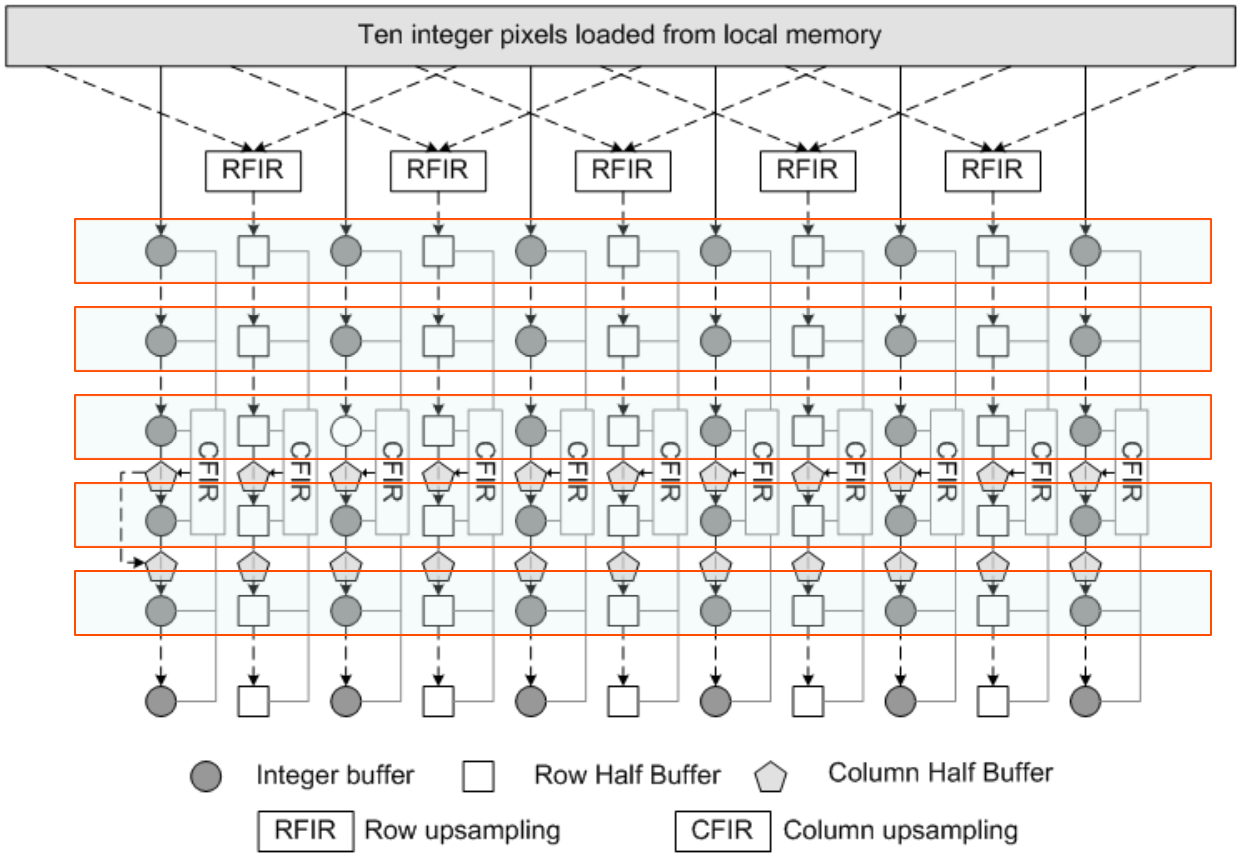
- Regular register files require 5 transfers per op
- Wasted energy in instruction fetch and register file

Augment register files with a custom shift register

- Parallel access of entries to create custom FIR arithmetic unit
- Result dissipates 1/30th of energy of traditional approach

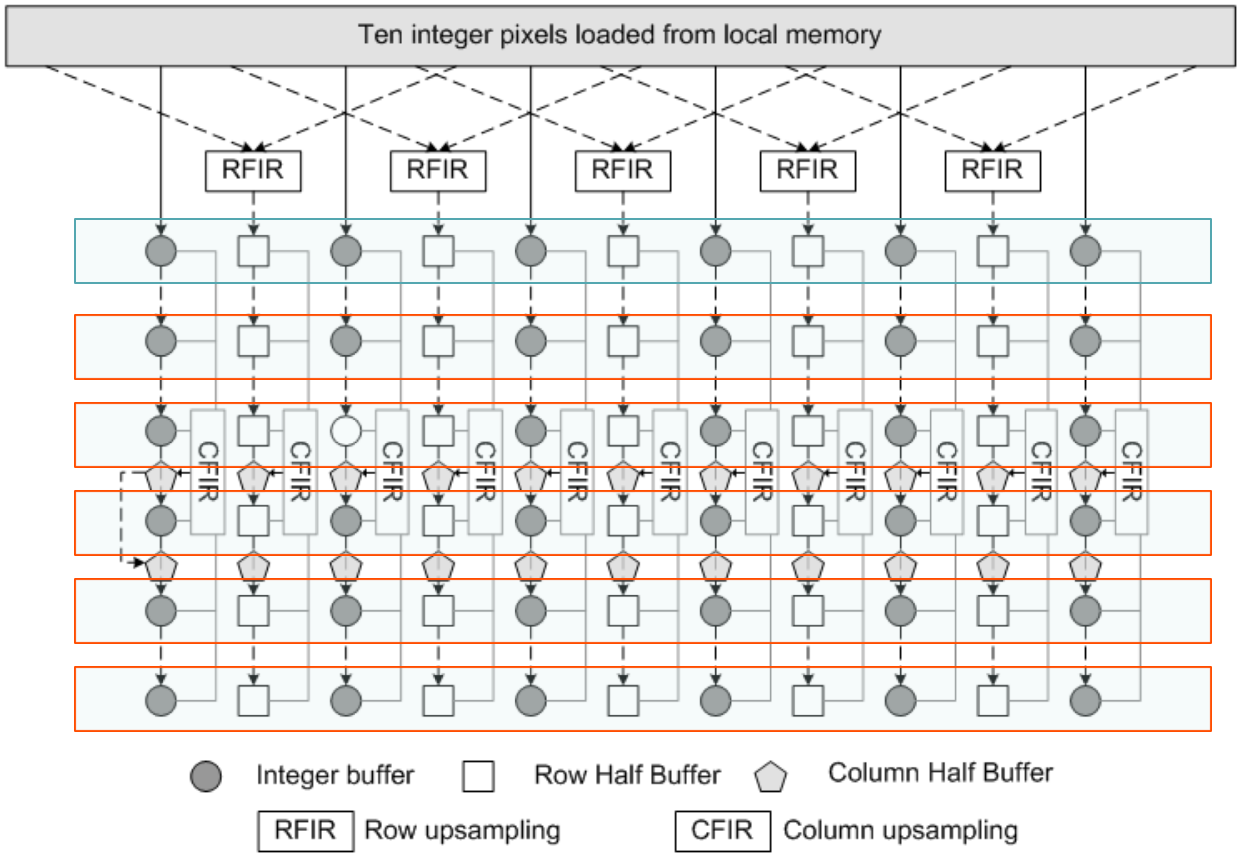
Custom FME

Custom upsampling datapath



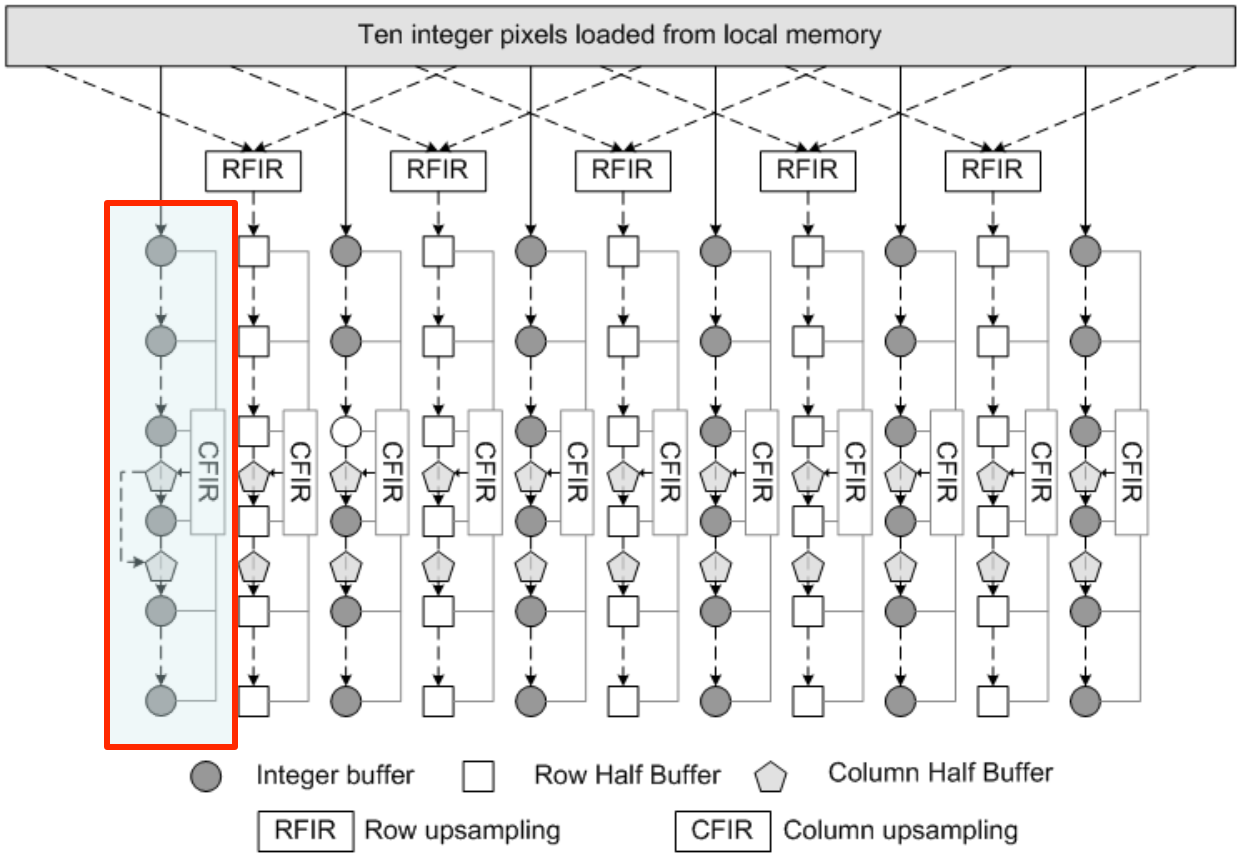
Custom FME

Custom upsampling datapath



Custom FME

Custom upsampling datapath



List Of Other Magic Instructions

Hadamard/DCT

- Matrix transpose unit
- Operation fusion with no limitation on number of operands

Intra Prediction

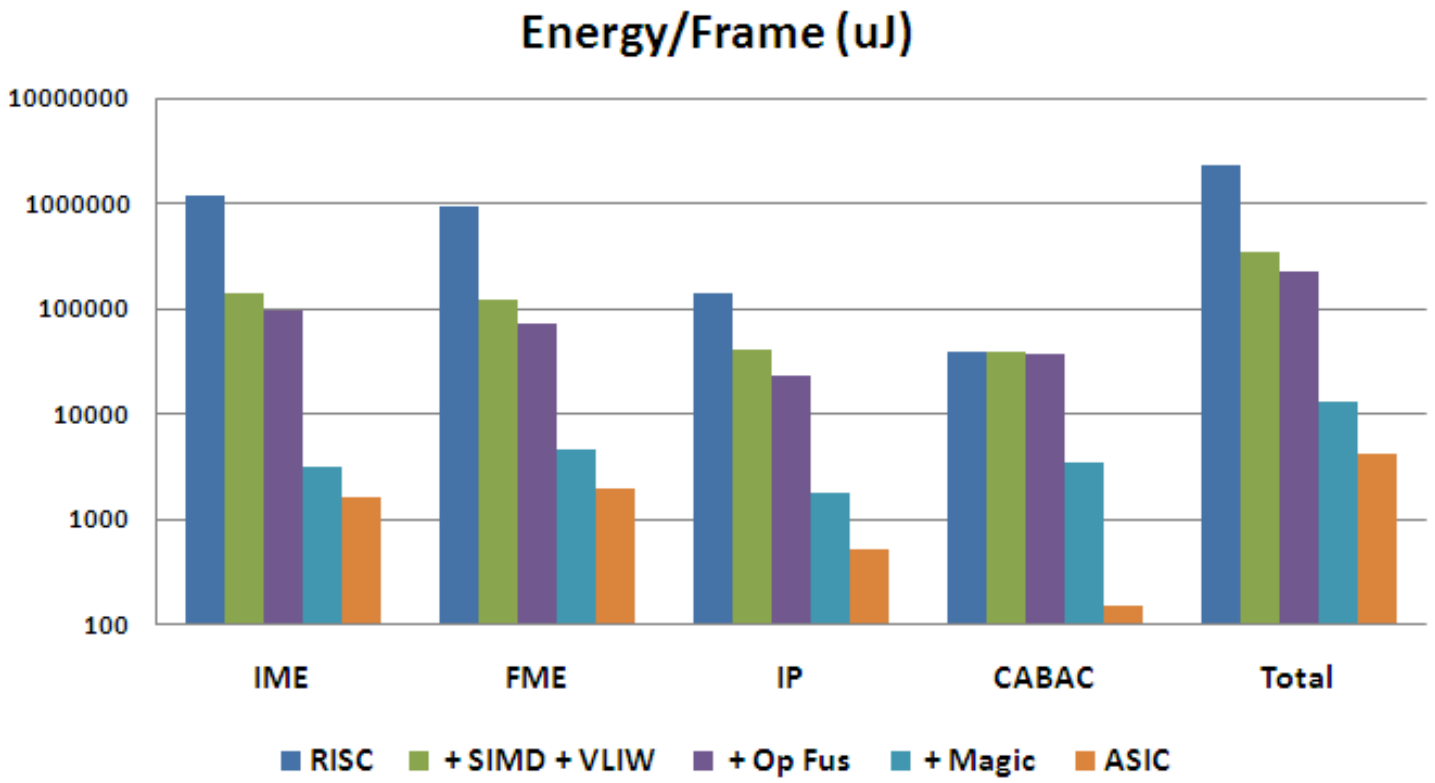
- Customized interconnections for different prediction modes

CABAC

- FIFO structures in binarization module
- Fundamentally different computation fused with no restrictions

Not many were needed

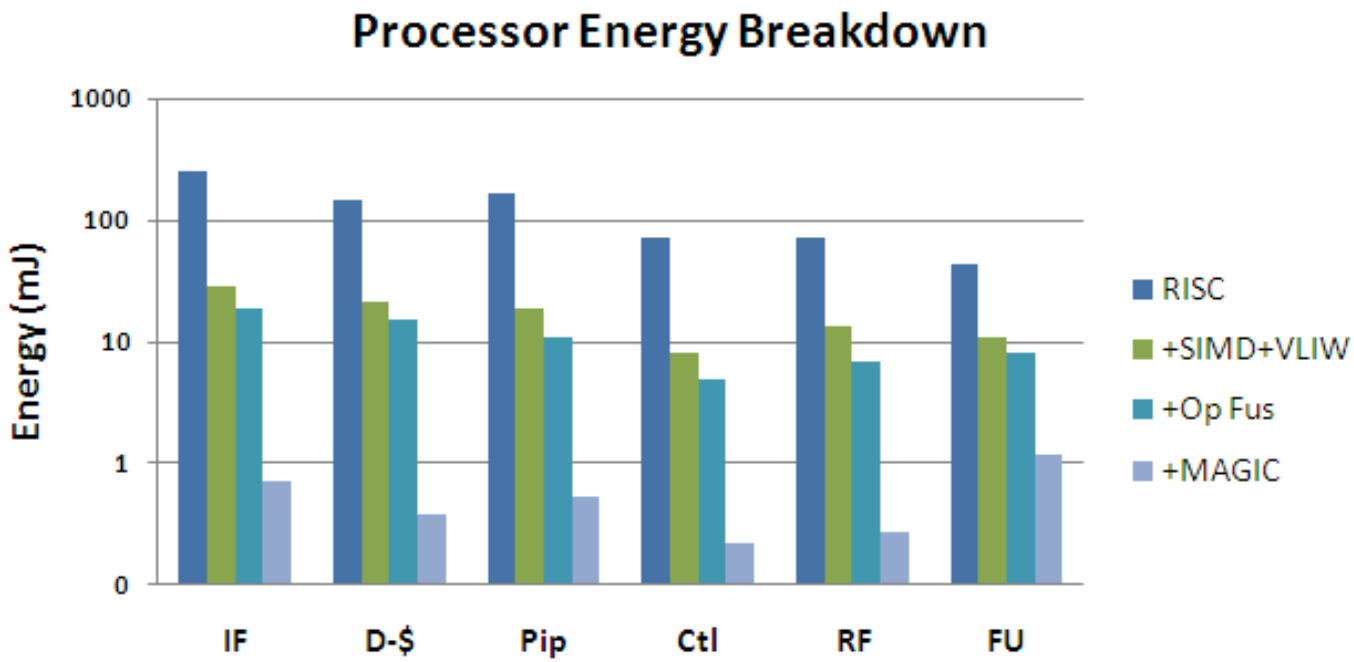
Magic Instructions - Energy



Efficiency orders of magnitude better than GP

Within 3X of ASIC energy efficiency

Magic instructions - Results



Over 35% energy now in ALU

- Overheads are well-amortized – *up to 256 ops / instruction*
- More data re-use within the data-path

Most of the code involves magic instructions

Magic Instructions Summary

Optimization strategy similar across all algorithms

- Closely couple data storage and data path structures
- Maximize data reuse inside the datapath

Commonly used hardware structures and techniques

- Shift registers with parallel access to internal values
- Direct computation of the desired output
  - Eliminate the generation (and storage) of intermediate results

Hundreds of extremely low-power ops per instruction

Works well for both data parallel and sequential algorithms

Conclusion

Many operations are very simple and low energy

- They SIMD/Vector parallelize well, but overheads still dominate
- To get ASIC efficiencies, need 100s ops/instruction
  - Specialized hardware/memory

Building ASIC hardware in a processor worked well

- Easier than building an ASIC, since it was incremental
- Start with strong software development environment
  - Add and debug only the hardware you need

Efficient hardware requires customization

- We should make doing chip customization feasible
- And that means we should design chip generators and not chips