Understanding Sources of Inefficiency in General-Purpose Chips

Hameed, Rehan, et al.

PRESENTED BY:

XIAOMING GUO

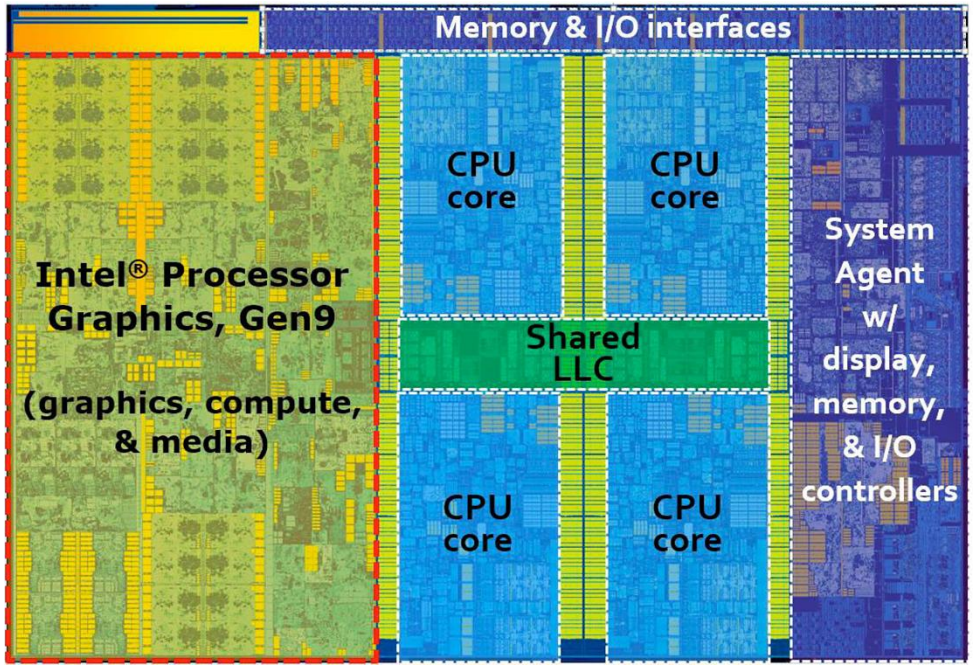
SIJIA HE

Outline

- Motivation
- H.264 Basics
- Key ideas
- Implementation & Evaluation
- Summary
- Conclusion
- Discussion

We Love General-Purpose Chips…

Intel Core i7-6700K



https://www.techpowerup.com/215333/intel-skylake-die-layout-detailed.html

Runs a broad range of applications!

14nm lithography

4GHz frequency

4 cores/8 threads

64KB L1 cache/core

256KB L2 cache/core

8MB LLC

But……

General-purpose chips are **Inefficient** for specific tasks

Highly optimized 2.8GHz Intel Pentium 4 implementation of H.264 encoder vs. ASIC

	Performance (fps)	Area (mm^2)	Energy/frame (mJ)
Intel (720×480 SD)	30	122	742
Intel (1280×720 HD)	11	122	2023
ASIC (1280×720 HD)	30	8	4

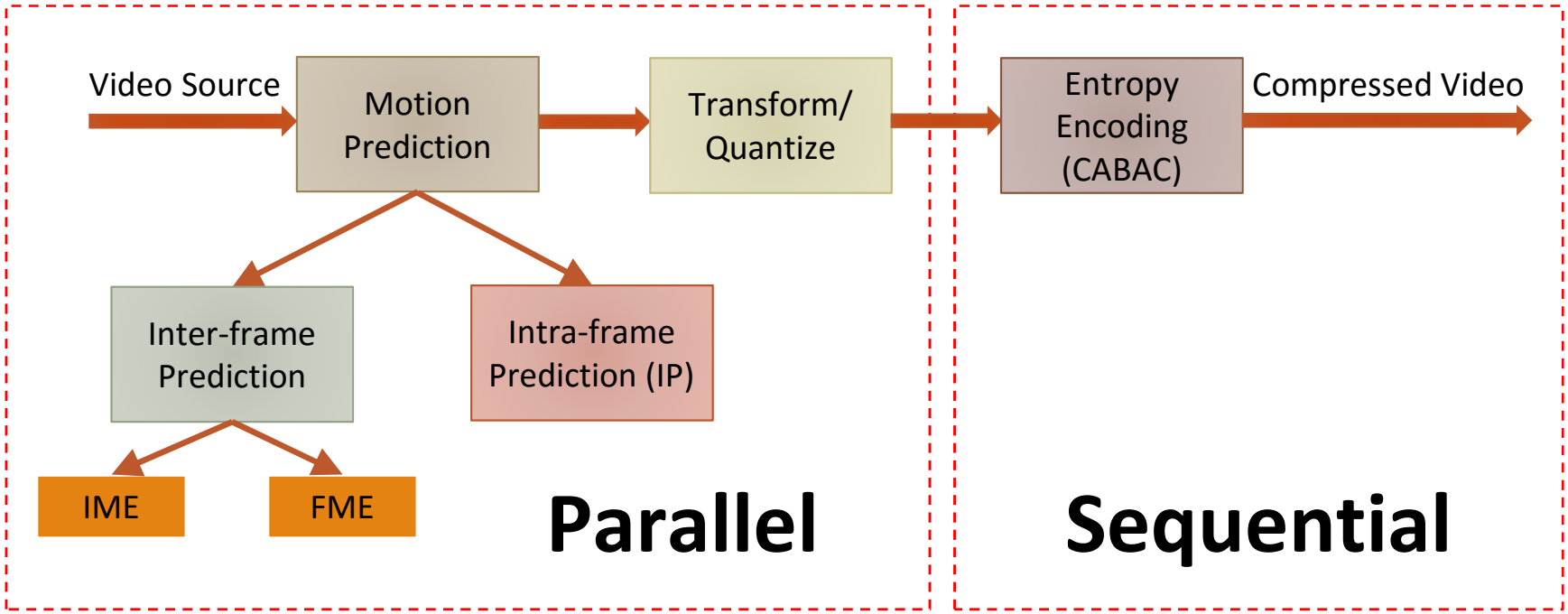
GP is 500x less energy efficient than ASIC in H.264 encoding

Outline

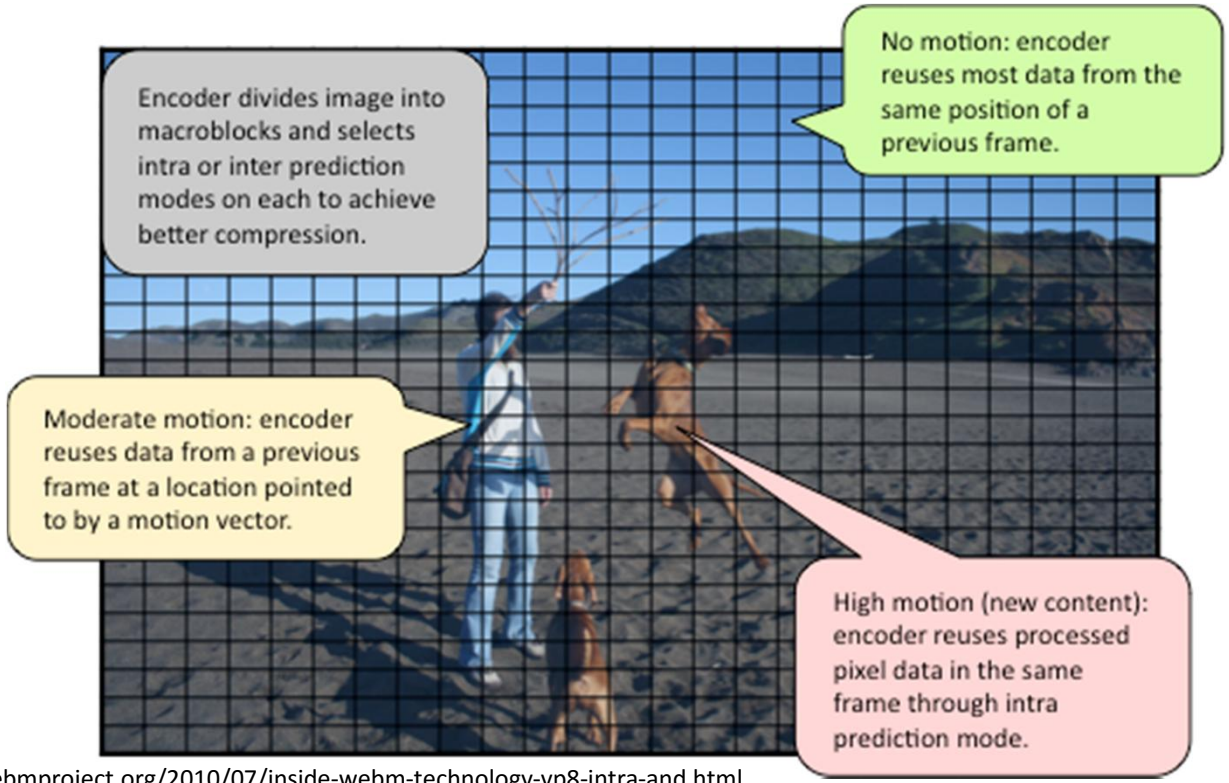
- Introduction
- H.264 Basics
- Key ideas
- Implementation & Evaluation
- Summary
- Conclusion
- Discussion

H.264 Basics

One of the most commonly used video coding format



H.264 Basics: An Example



http://blog.webmproject.org/2010/07/inside-webm-technology-vp8-intra-and.html

Outline

- Introduction
- H.264 Basics
- Key ideas
- Implementation & Evaluation
- Summary
- Conclusion
- Discussion

Key Ideas

1. Use an Tensilica-based, extensible CMP system as the baseline

2. Explore customizations of the baseline that reduce overheads to transform the baseline into an ASIC-like H.264 encoder

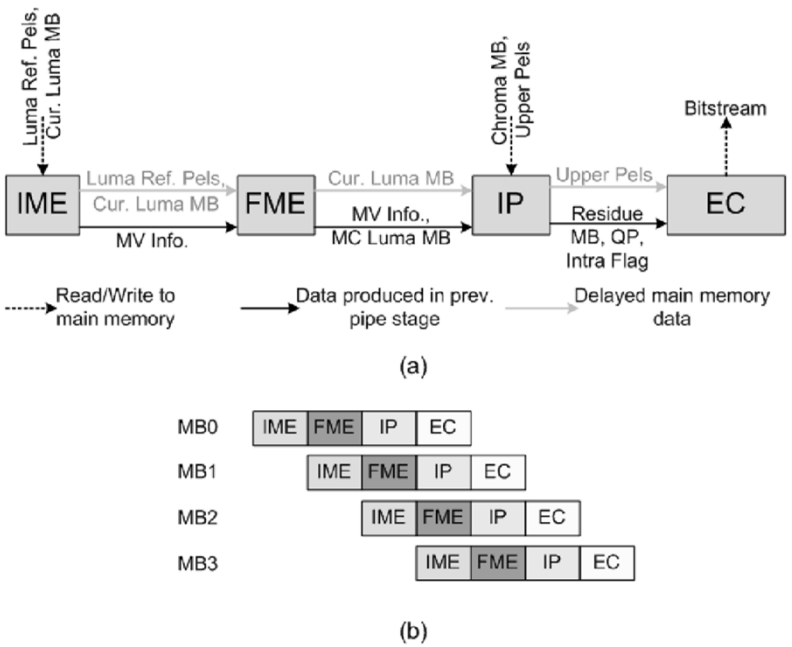
3. Study how the overhead is reduced

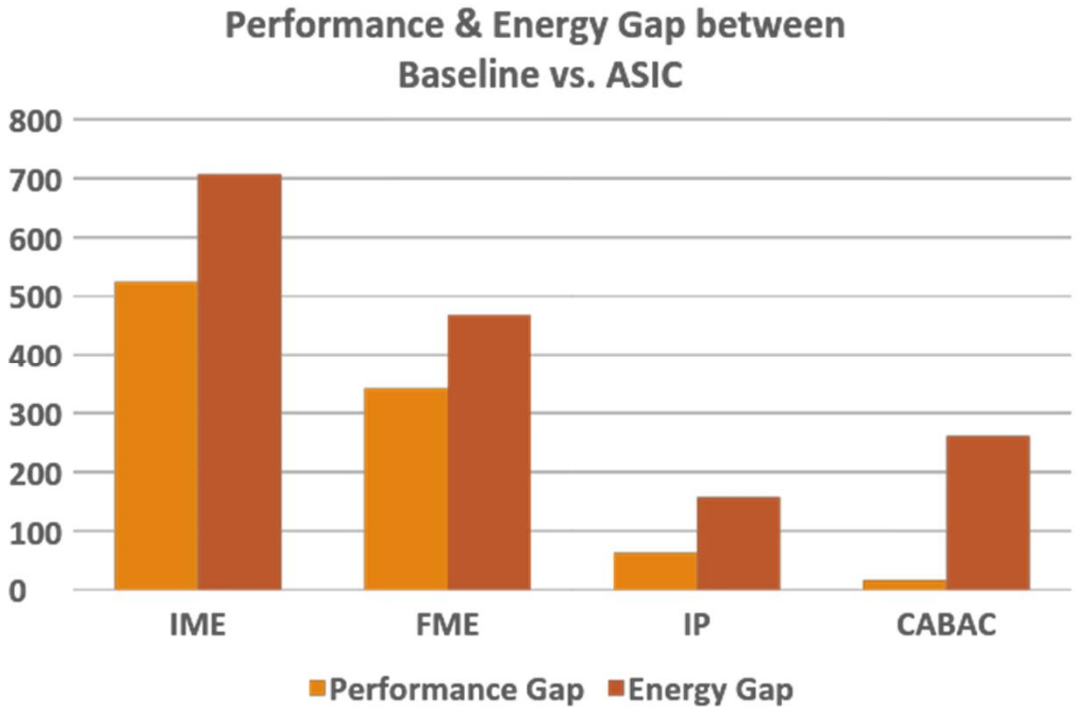
4. Understand the source of inefficiency in GP

Outline

- Introduction
- H.264 Basics
- Key ideas
- Implementation & Evaluation
- Summary
- Conclusion
- Discussion

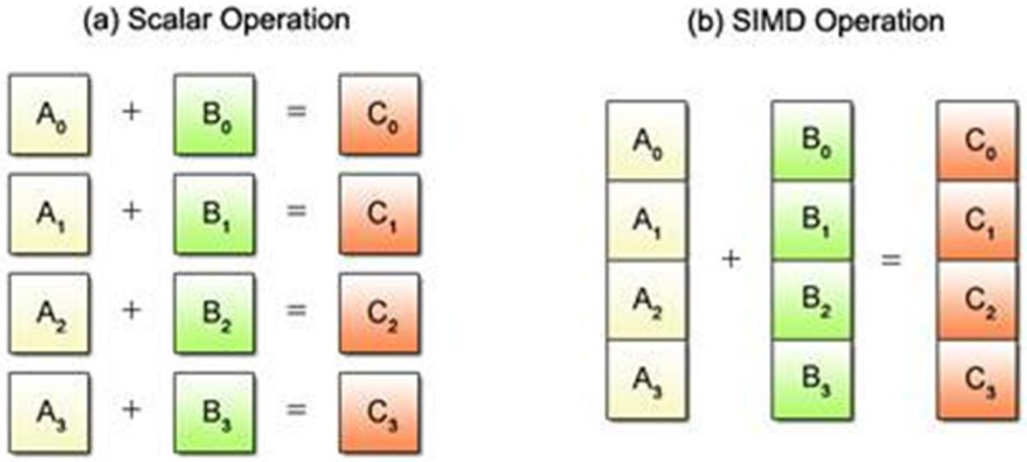
Implementation - Baseline





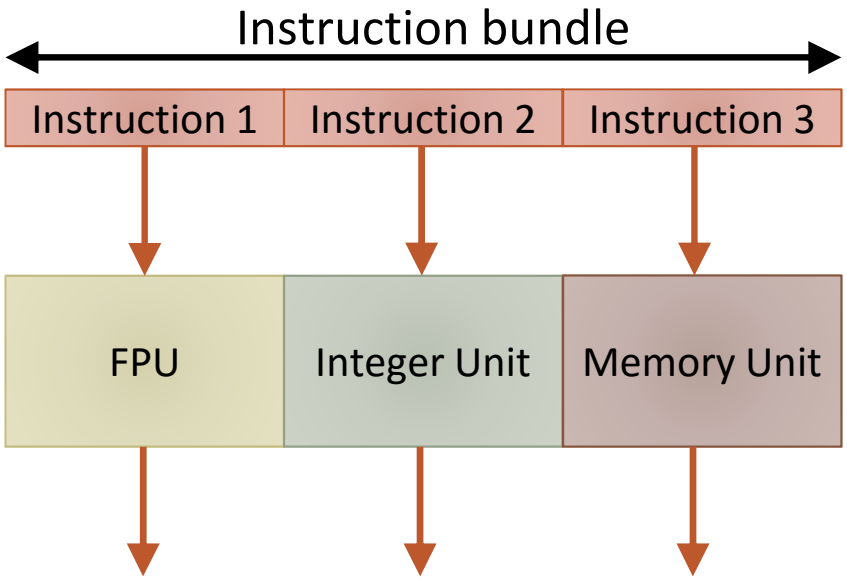
Implementation – Customization Step 1

Data Level Parallelism:
Up to 18-way **SIMD**

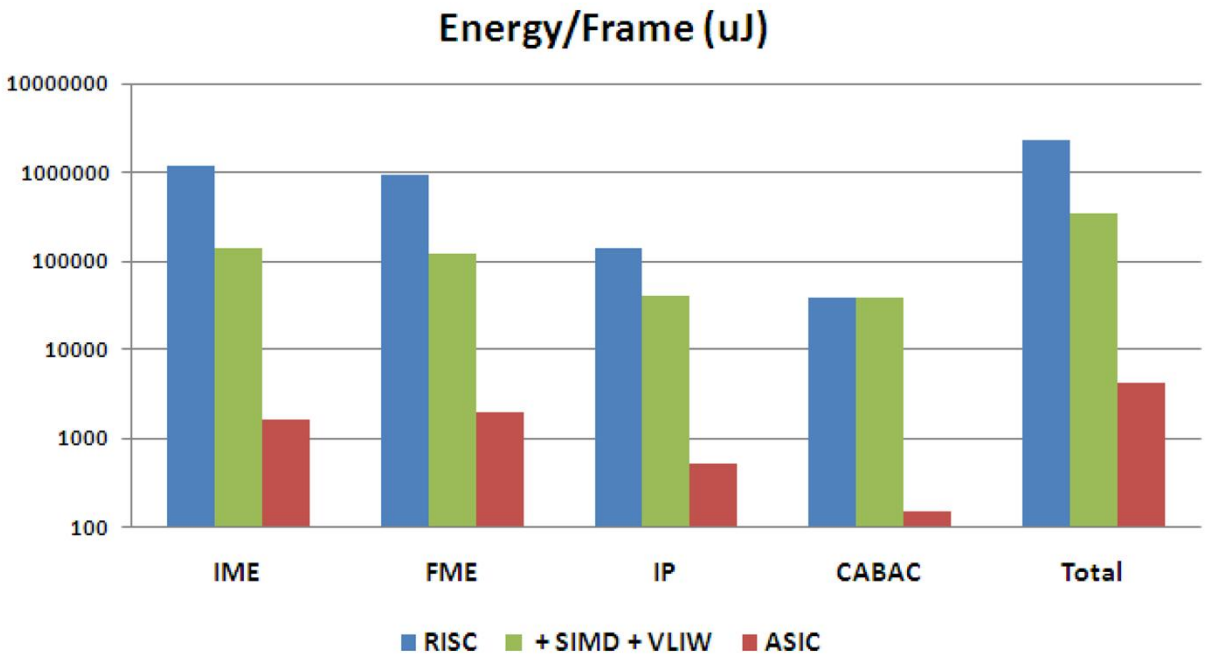


https://www.kernel.org/pub/linux/kernel/people/geoff/cell/ps3-linux-docs/CellProgrammingTutorial/BasicsOfSIMDProgramming.html

Instruction Level Parallelism:
Up to 3-slot **VLIW**



Evaluation – Customization Step 1



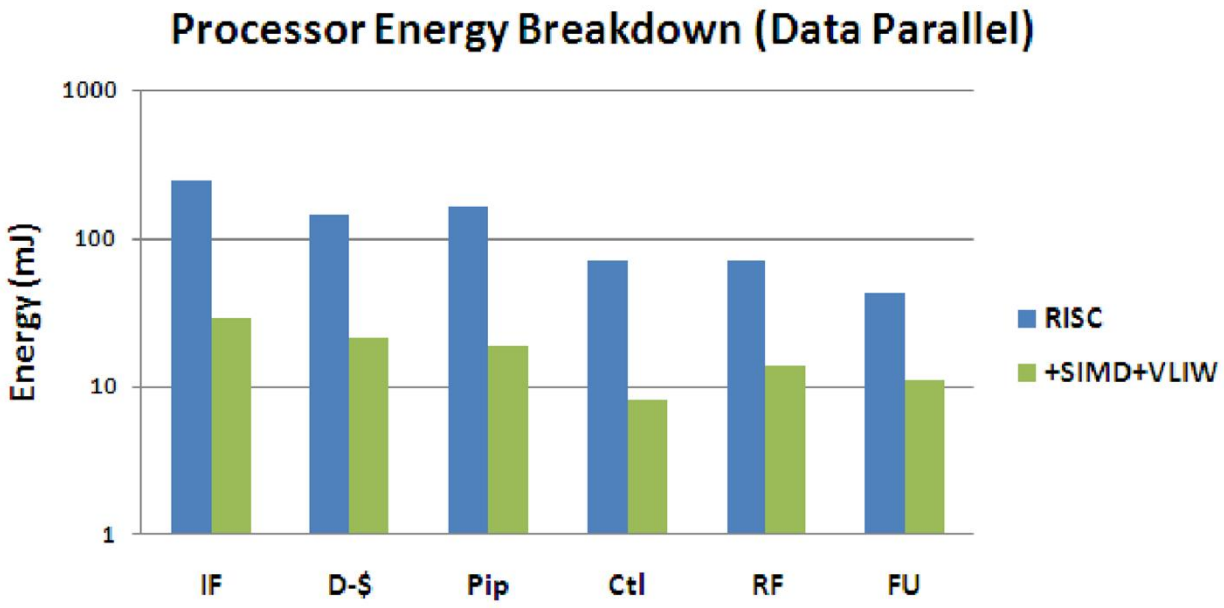
Speedup: 9.2x

Good: An order of magnitude increase in performance and energy efficiency

Bad: Still two orders of magnitude worse than ASIC

Taken from the presentation slides for this paper on 2010 ISCA

Evaluation – Customization Step 1



Taken from the presentation slides for this paper on 2010 ISCA

Good: Function unit becomes more energy efficient

Bad: Overheads from other operations are still significant. FU only accounts for 10% in all the energy consumed.

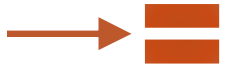
Implementation – Customization Step 2

Operation Fusion: Combine several instructions into one

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

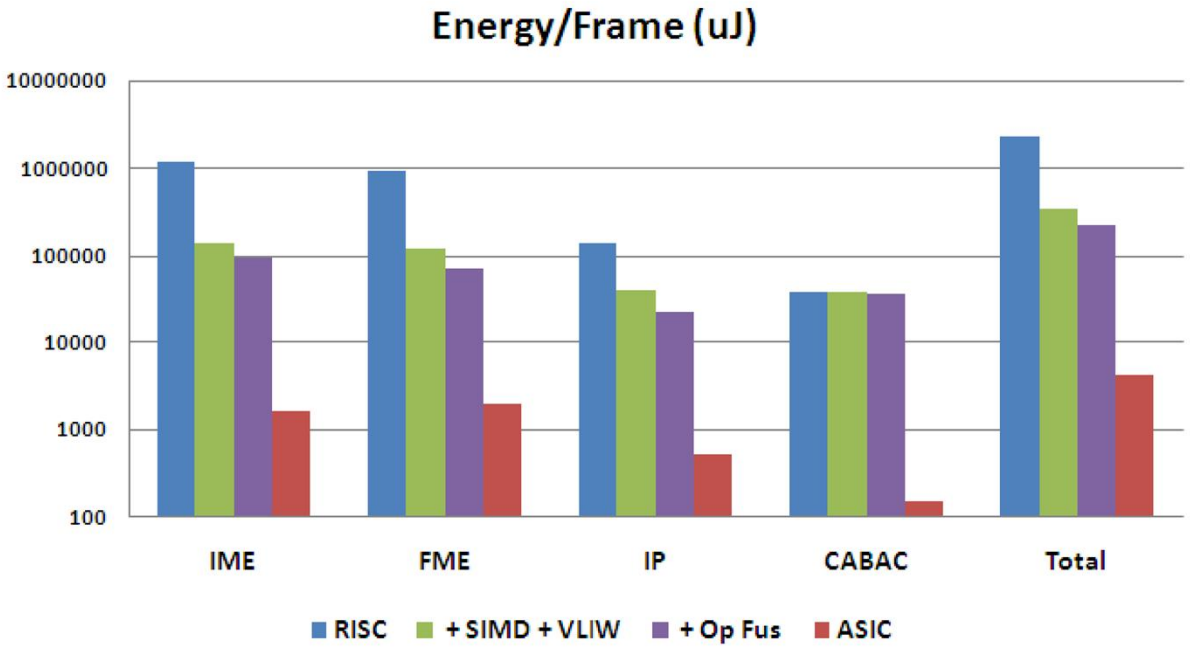
$$x_n = 20(x_0 + x_1) - 5(x_{-1} + x_2) + (x_{-2} + x_3)$$

Pseudo-code:
acc = 0;
acc = Addshft (acc, x_0, x_1, 20);
acc = Addshft (acc, x_0, x_1, -5);
acc = Addshft (acc, x_0, x_1, 1);
x_n = Sat(acc);



acc = acc + 20(x_0 + x_1):
temp1 = 20 * x_0
temp2 = 20 * x_1
temp3 = temp1 + temp2
acc = acc + temp3

Evaluation – Customization Step 2



Speedup: 15.7x (baseline)
Speedup: 1.7x (SIMD+VLIW)

Good: Compiler can do operation fusion. So it is "free".

Bad: No significant change in performance and energy efficiency compared to SIMD + VLIW

Taken from the presentation slides for this paper on 2010 ISCA

Evaluation – Observations So Far…

Exploring data and instruction level parallelism helps

- 10x improvement in both performance and energy efficiency

The inefficiency is dominated by data movement

- FU only takes 10% of total energy even with SIMD + VLIW

Need large computation/low communication

- Specific hardware to perform 100s operations with 1 instruction

Implementation – Customization Step 3

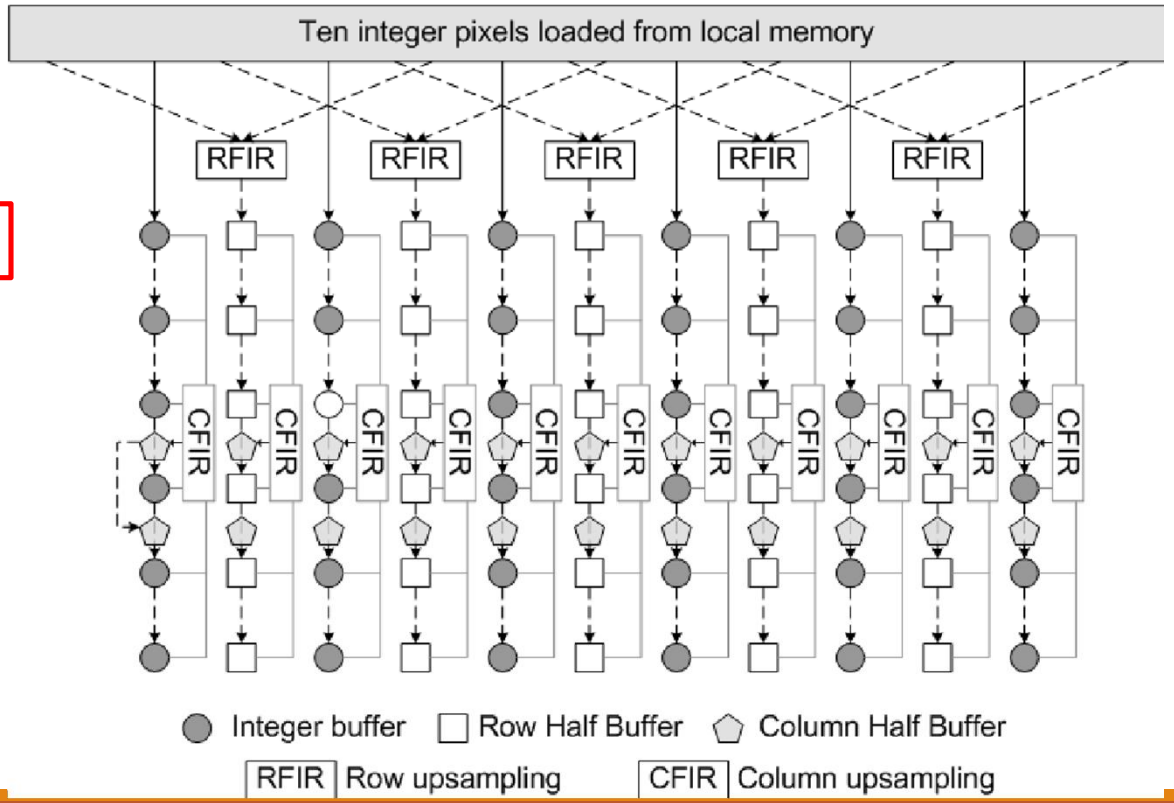
Special Hardware for FME

Cycle 1:

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

Cycle 2:

$$x_n = \boxed{x_{-1} - 5x_0 + 20x_1 + 20x_2 - 5x_3} + x_4$$



Implementation – Customization Step 3

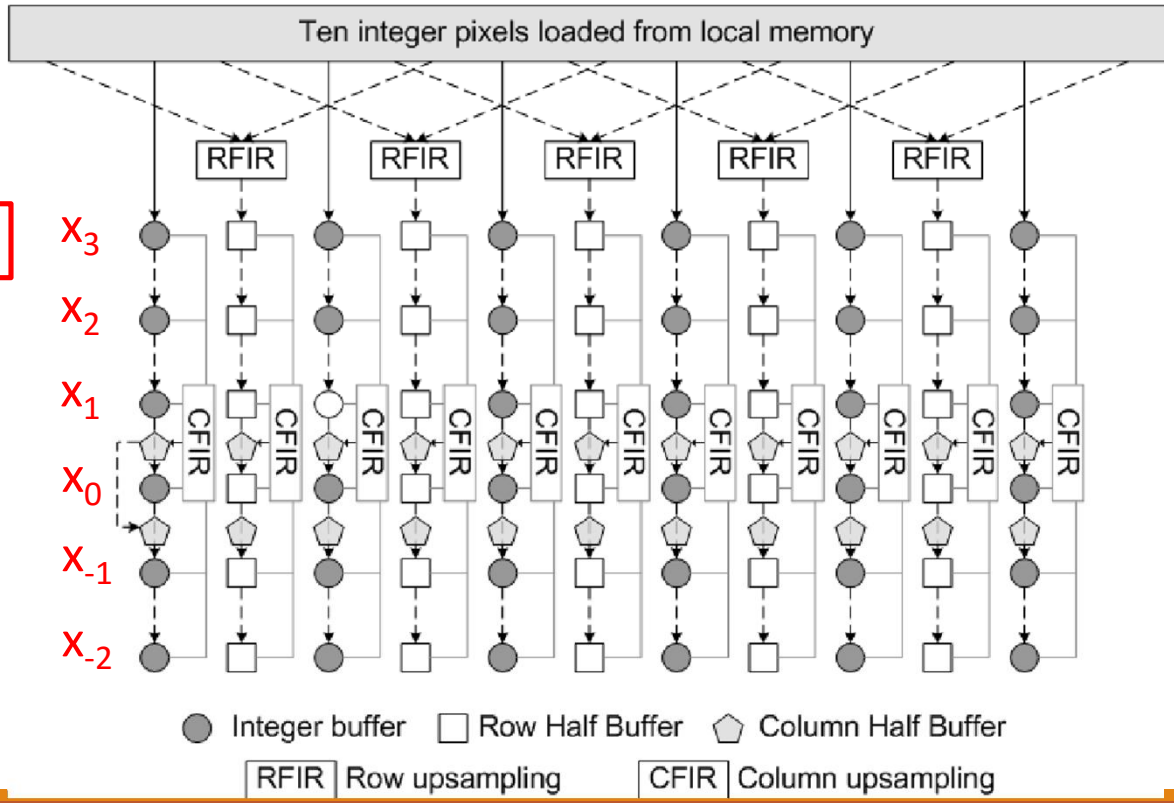
Special Hardware for FME

Cycle 1:

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

Cycle 2:

$$x_n = \boxed{x_{-1} - 5x_0 + 20x_1 + 20x_2 - 5x_3} + x_4$$



Implementation – Customization Step 3

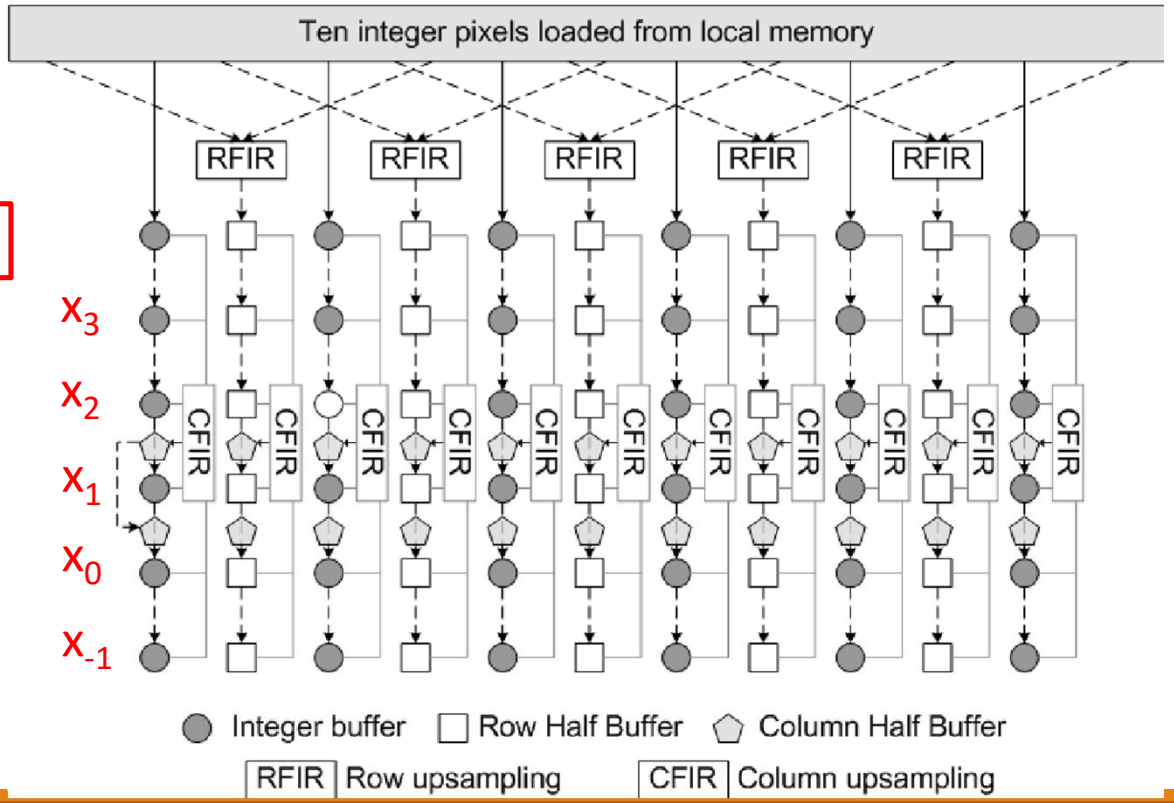
Special Hardware for FME

Cycle 1:

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

Cycle 2:

$$x_n = \boxed{x_{-1} - 5x_0 + 20x_1 + 20x_2 - 5x_3} + x_4$$



Implementation – Customization Step 3

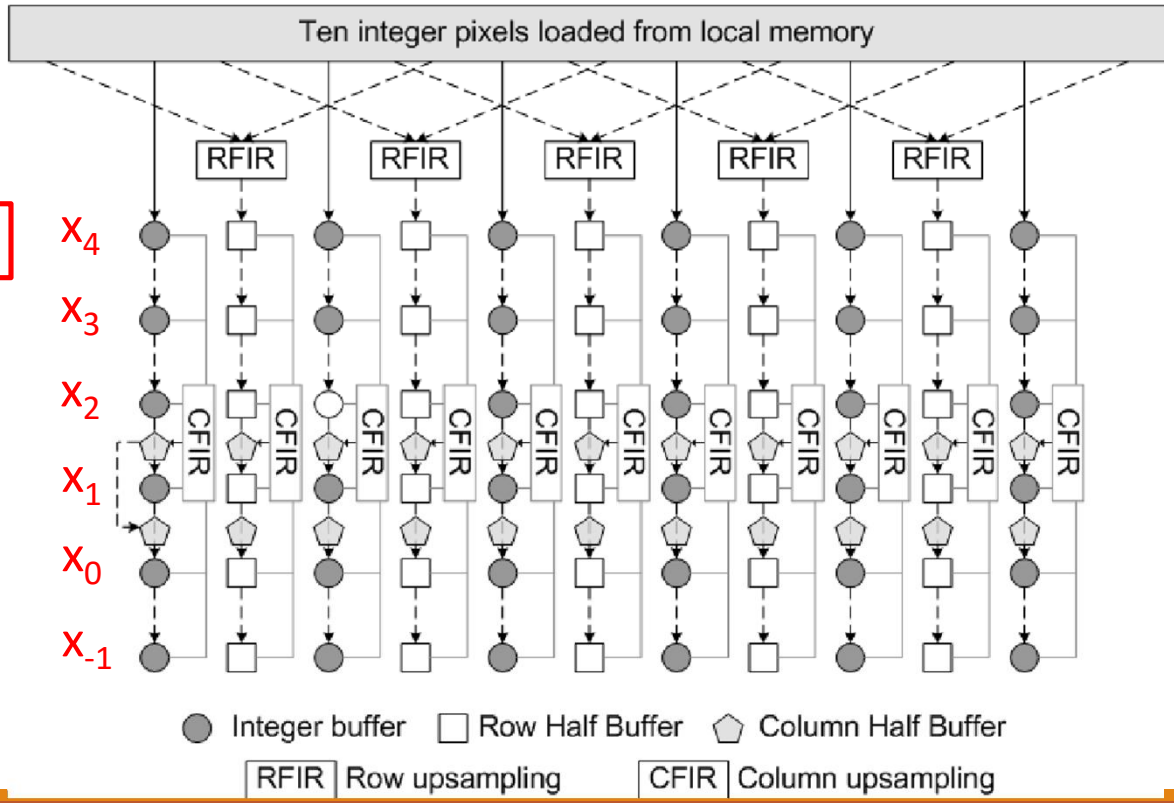
Special Hardware for FME

Cycle 1:

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

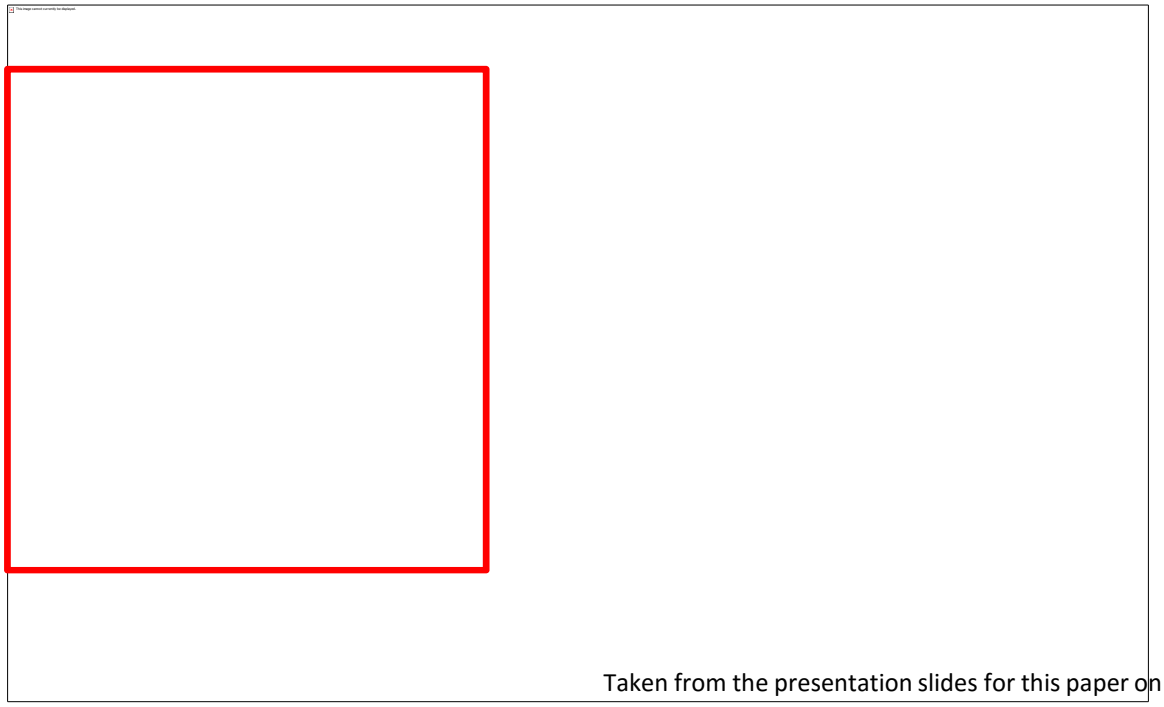
Cycle 2:

$$x_n = \boxed{x_{-1} - 5x_0 + 20x_1 + 20x_2 - 5x_3} + x_4$$



Implementation – Customization Step 3

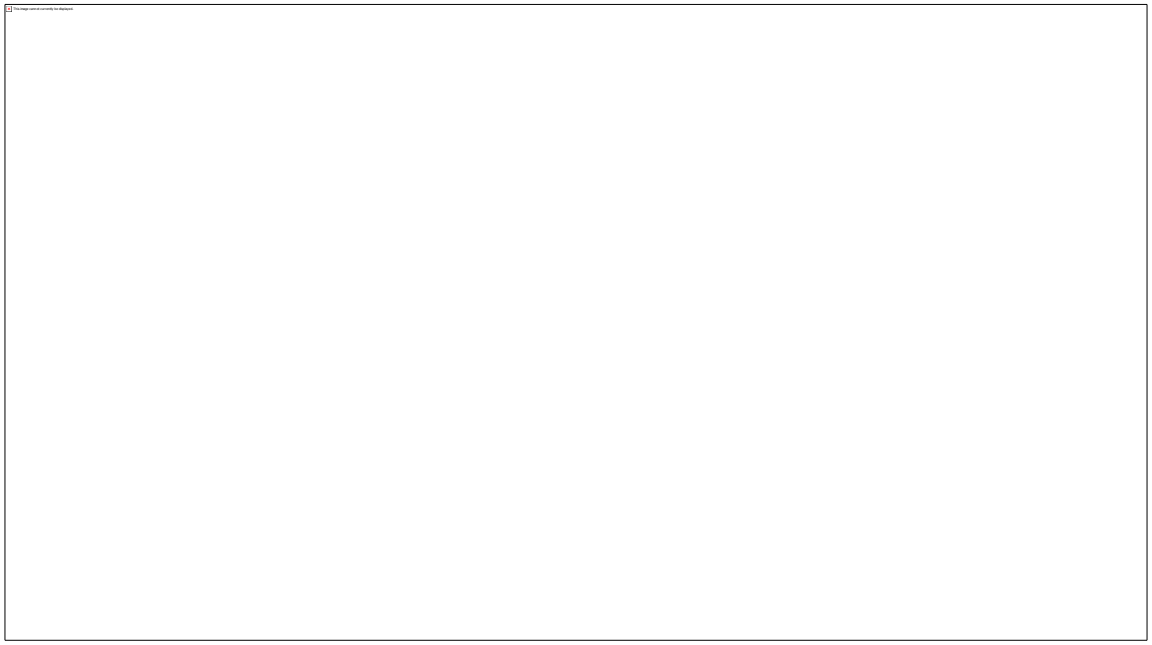
Special Hardware for IME



Taken from the presentation slides for this paper on 2010 ISCA

- SAD: Sum of Absolute Difference
- 256 SAD / operation
- Pixels blocks in red box can shift down and right, maximizing data re-use.
- Minimize bits fetched/operation

Evaluation – Customization Step 3



Taken from the presentation slides for this paper on 2010 ISCA

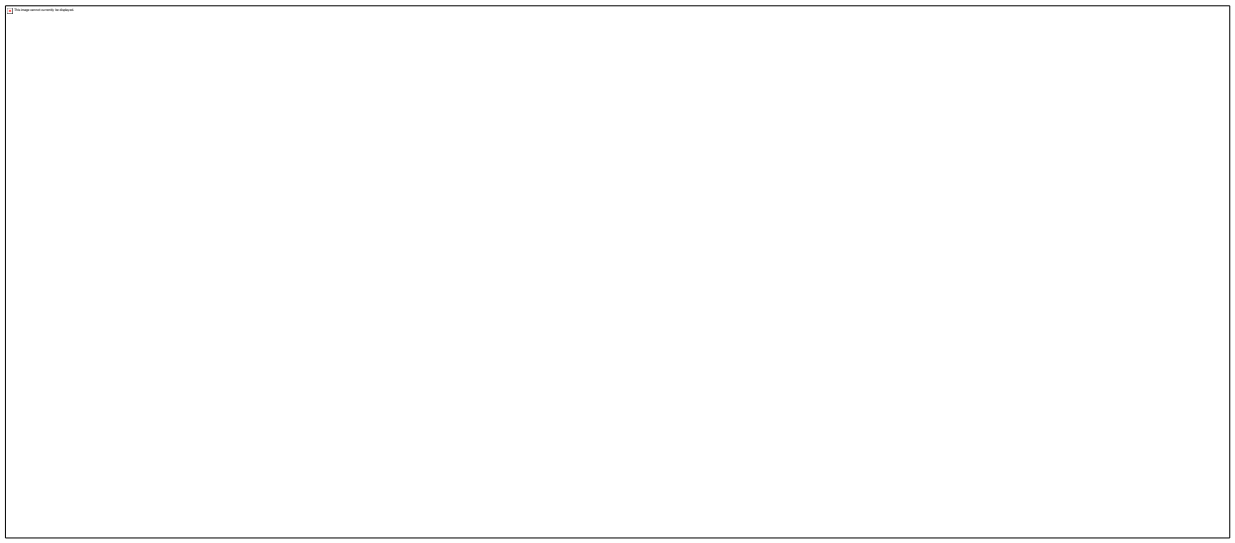
Speedup: 256x (baseline)

Good: Orders of magnitude faster than GP. Energy efficiency within 3x of ASIC.

Bad: Maybe very difficult to implement in GP. Requires customized data storage, function unit and instruction.

Evaluation – Customization Step 3

Processor Energy Breakdown



Taken from the presentation slides for this paper on 2010 ISCA

Good: Over 35% of the energy is now in FU. More data re-use, lowering total energy.

Bad: Most of the code involves “magic” instructions

Outline

- Introduction
- H.264 Basics
- Key ideas
- Implementation & Evaluation
- Summary
- Conclusion
- Discussion

Summary

- Basic operations only take hundreds femtojoule/op in ASIC
- Instruction/data fetch, pipeline registers, control, etc. take 140pJ/op
- Even with SIMD, VLIW and operation fusion, the gap is hard to close
- Need “magic instruction” that can perform **100s operations** each time
- Can be done by coupling customized data storage and data path
- Example: using shift registers to enable data reuse and parallel access

Outline

- Introduction
- H.264 Basics
- Key ideas
- Implementation & Evaluation
- Summary
- Conclusion
- Discussion

Conclusion

➢Achieving ASIC-like energy efficiency is difficult due to the large overheads in fetch, pipeline registers, control, etc.

➢Need customized hardware that fits in processor framework that can execute 100s of operations/instruction.

➢Need tools that enable designers to create customized hardware easily to reduce the cost. (Bridging Moore's Law performance gap is more about **"How Much"** does it cost – Prof. Todd Austin)

Questions?

Discussion

1. The paper only studied the gap between general purpose chips and H.264 ASIC. Do the results also apply to other application?

1. The authors introduced “magic” instructions that can perform 100s operations to achieve ASIC-like efficiency. Is it realistic to have those instructions in a real design?