7. Optimization

DR. B. I. MORSHED

ASSOCIATE PROFESSOR
COMPUTER SCIENCE
TEXAS TECH UNIVERSITY

Embedded Systems/Cyber Physical Systems

CS 4380 / CS 5331

Chapter Outline

- Performance of ES
- Pareto Point
- Pareto set
- Pareto front
- Optimization approaches
- Power optimization
- Other optimizations

How to evaluate designs with multiple criteria?

In practice, many different criteria are relevant for evaluating designs

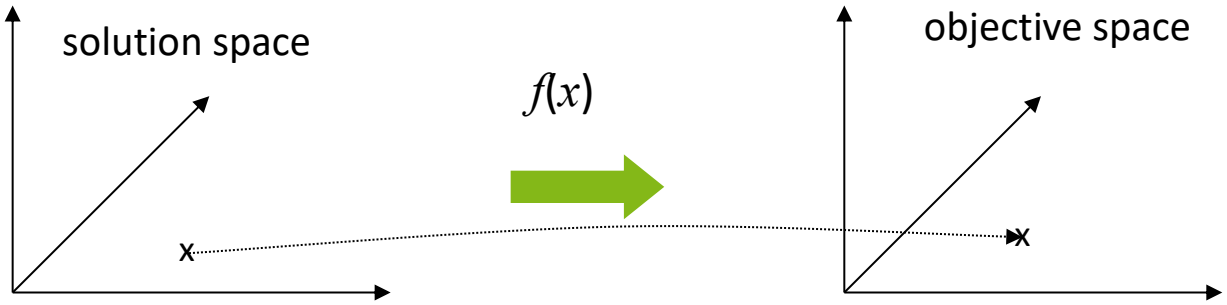
Different design objectives/criteria are relevant:

- Average performance
- Worst case performance
- Energy/power consumption
- Thermal behavior
- Reliability
- Electromagnetic compatibility
- Numeric precision, Testability, Cost, weight, robustness, usability, extendibility, security, safety, environmental ..

How to compare designs? Some designs “better” than others – but why and how?

Solution Space & Objective Space

- Let X: m-dimensional **solution space** for the design problem.
  Example: dimensions correspond to # of processors, size of memories, type and width of busses etc.
- Let F: n-dimensional **objective space** for the design problem.
  Example: dimensions correspond to speed, cost, power consumption, size, weight, reliability, …
- Let $f(x)=(f_1(x),\ldots,f_n(x))$ where $x \in X$ be an **objective function.**
  We assume that we are using $f(x)$ for evaluating designs.



Objectives and solutions

Given:

Goal:

Objectives: cost, latency, power consumption

Definitions for Pareto Analysis

- We assume that, for each objective, a total order (<) and the corresponding order (≤) are defined.
- **Definition**:
  Vector $u = (u_1, \ldots, u_n) \in F$ **dominates** vector $v = (v_1, \ldots, v_n) \in F$
  iff u is "better" than v with respect to one objective and not worse than v with respect to all other objectives:

$$\forall i \in \{1, \ldots, n\} : u_i \leq v_i$$

$$\wedge$$

$$\exists i \in \{1, \ldots, n\} : u_i < v_i$$

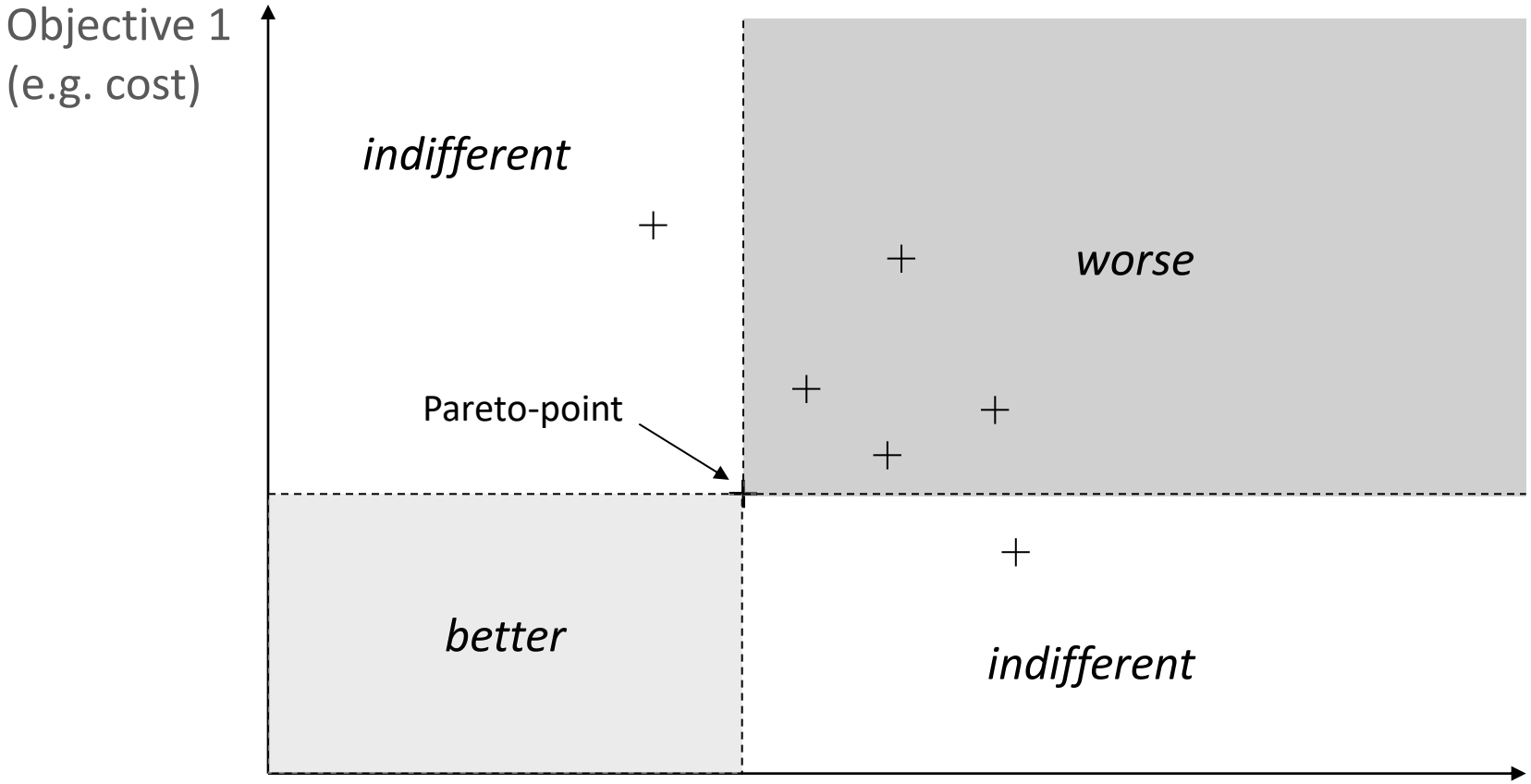
- **Definition:**
  Vector $u \in F$ is **indifferent** with respect to vector $v \in F$
  iff neither u dominates v nor v dominates u

Pareto-optimal and non-dominate

- A solution $x \in X$ is called **Pareto-optimal** with respect to X iff there is no solution $y \in X$ such that $u=f(x)$ is dominated by $v=f(y)$

- Let $S \subseteq F$ be a subset in objective space.
- **Definition**: v is called a **non-dominated solution** with respect to S iff v is not dominated by any element $\in S$.
- v is called **Pareto-optimal** iff v is non-dominated with respect to all solutions F.

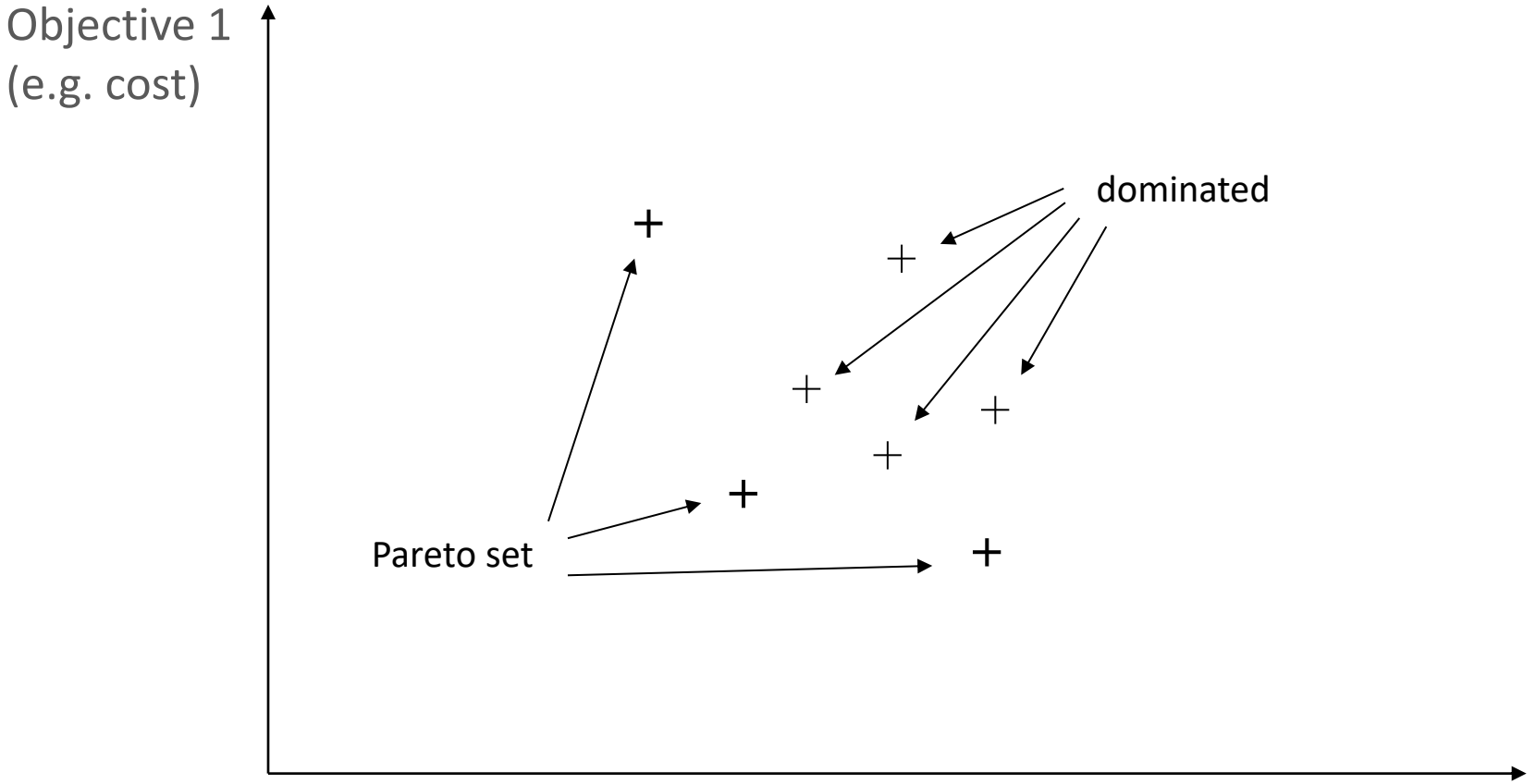
Pareto Points



(Assuming *minimization* of objective is better)

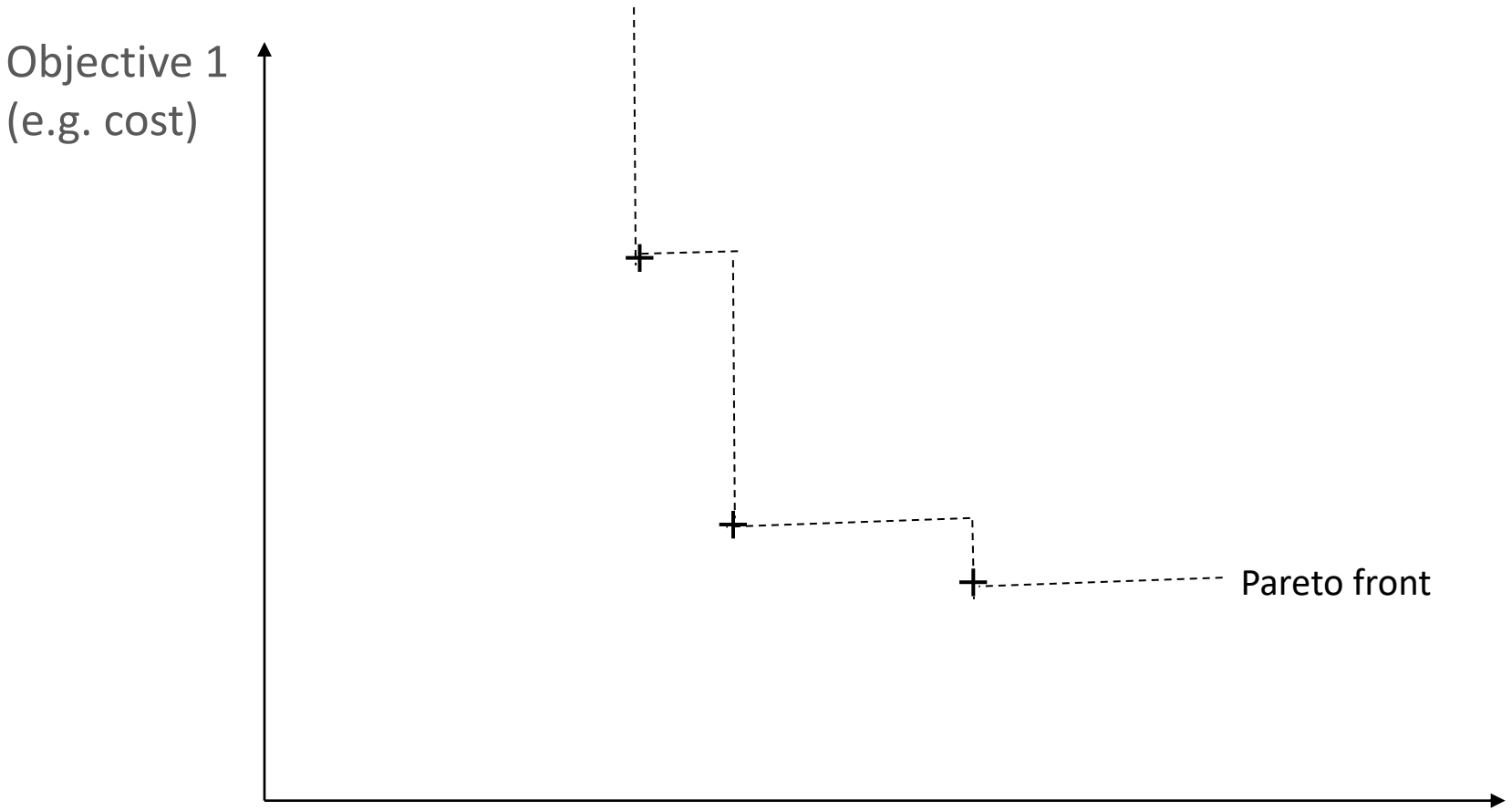
Pareto Set

Pareto set = set of all Pareto-optimal solutions



(Assuming *minimization* of objective is better)

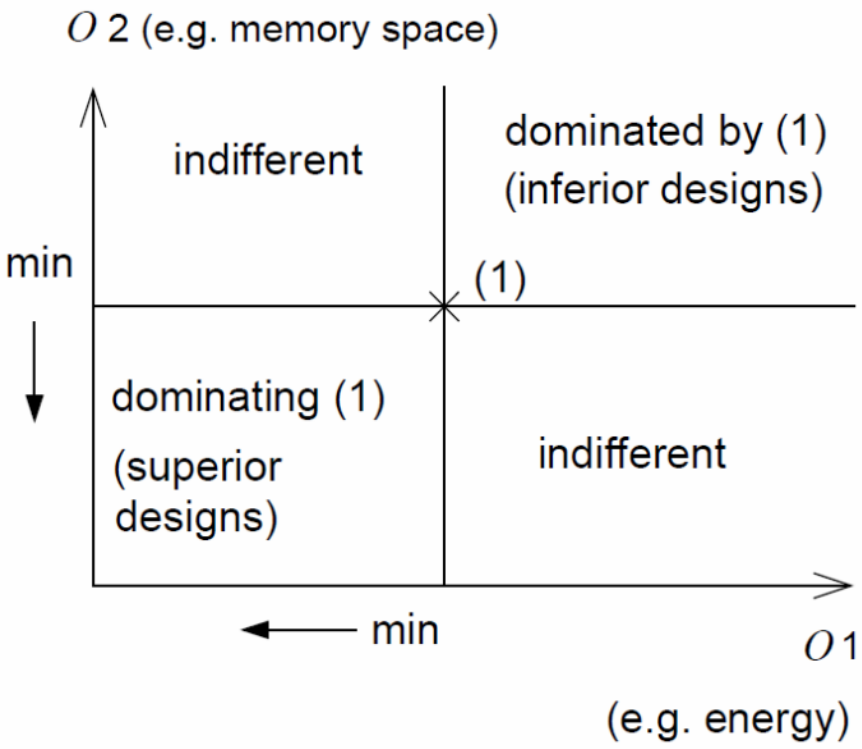
Pareto Front



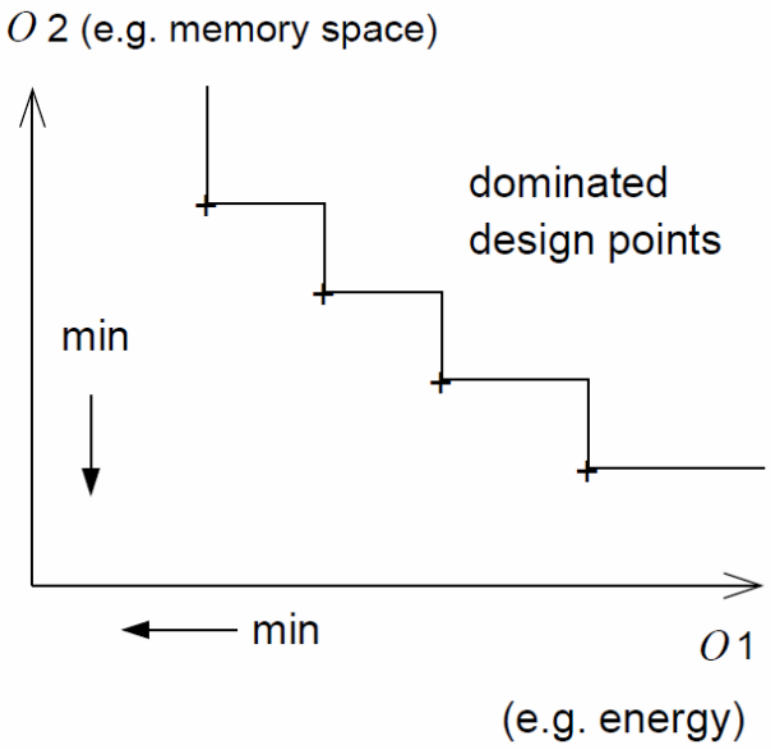
(Assuming *minimization* of objective is better)

Another example of Pareto Front

Pareto point



Pareto front



Optimization: Thermal

- Thermal models becoming increasingly important
  - since temperatures become more relevant due to increased performance, and
  - since temperatures affect
    - usability and
    - dependability.

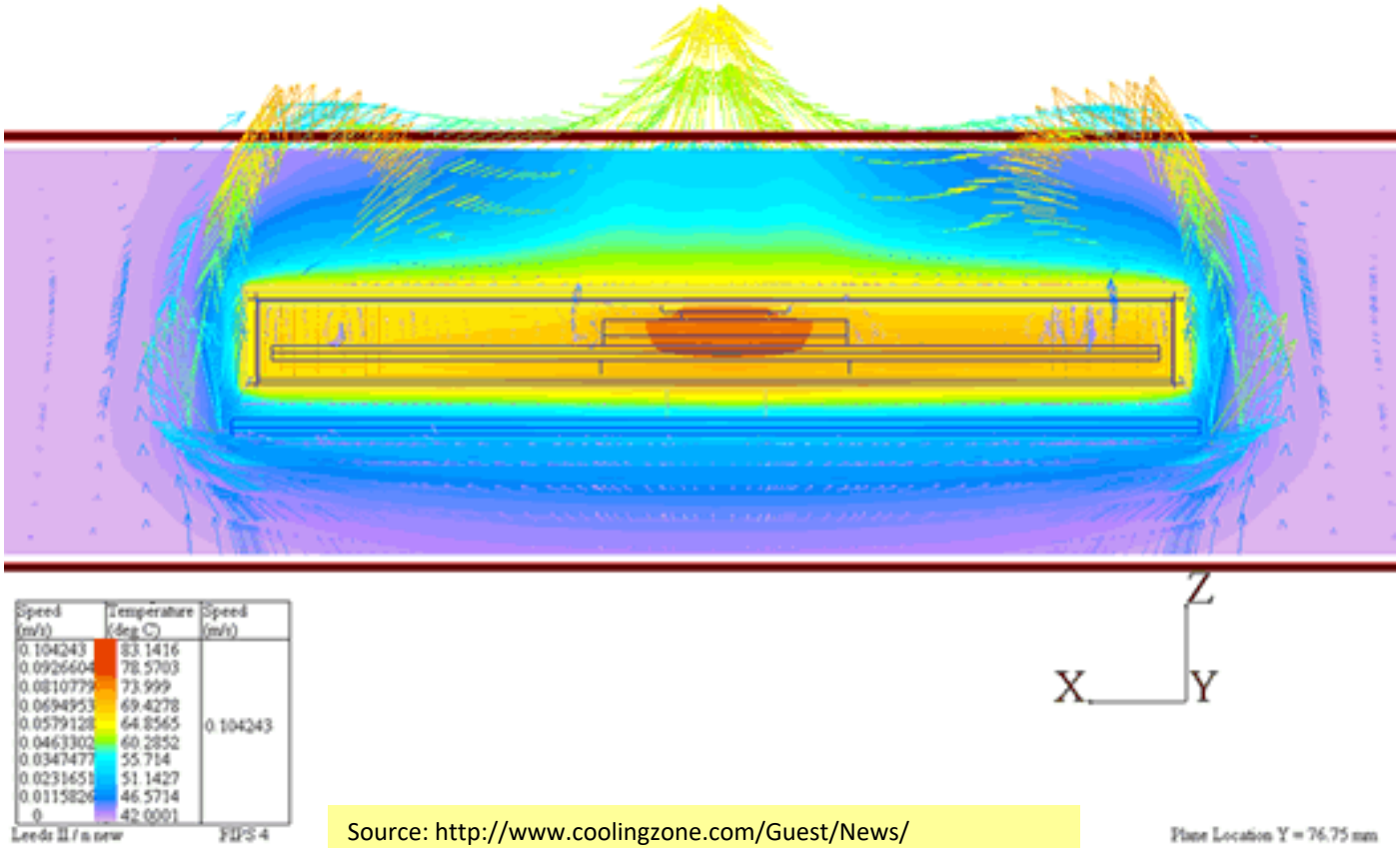
Thermal Model

- **Thermal conductance** reflects the amount of heat transferred through a plate (made from that material) of area A and thickness L when the temperatures at the opposite sides differ by one Kelvin.
- The reciprocal of thermal conductance is called **thermal resistance**.
- **Thermal resistances add up** like electrical resistances
- Masses storing heat correspond to **capacitors**

☞ Thermal modeling typically uses **equivalent electrical models** and employs well-known techniques for solving electrical network equations

Simulations on thermal models

Encapsulated cryptographic coprocessor:



Source: http://www.coolingzone.com/Guest/News/NL_JUN_2001/Campi/Jun_Campi_2001.html

Electro-magnetic compatibility (EMC)

Example: car engine controller

Red: high emission; Validation of EMC properties often done at the end of the design phase.

Source: http://intrage.insa-tlse.fr/~etienne/emccourse/what_for.html

Simulation Limitations

- Typically slower than the actual design.
  ☞ **Violations of timing constraints** likely if simulator is connected to the actual environment
- Simulations in the real environment may be **dangerous**
- There may be huge amounts of data and it may be impossible to simulate enough data in the available time.
- Most actual systems are too complex to allow simulating all possible cases (inputs).
  Simulations can help finding errors in designs,
  but they cannot guarantee the absence of errors.

Power Optimization

Extremely important for battery operated devices

Some factors effected:

- Hours of operation
- Thermal
- Usability
- Access

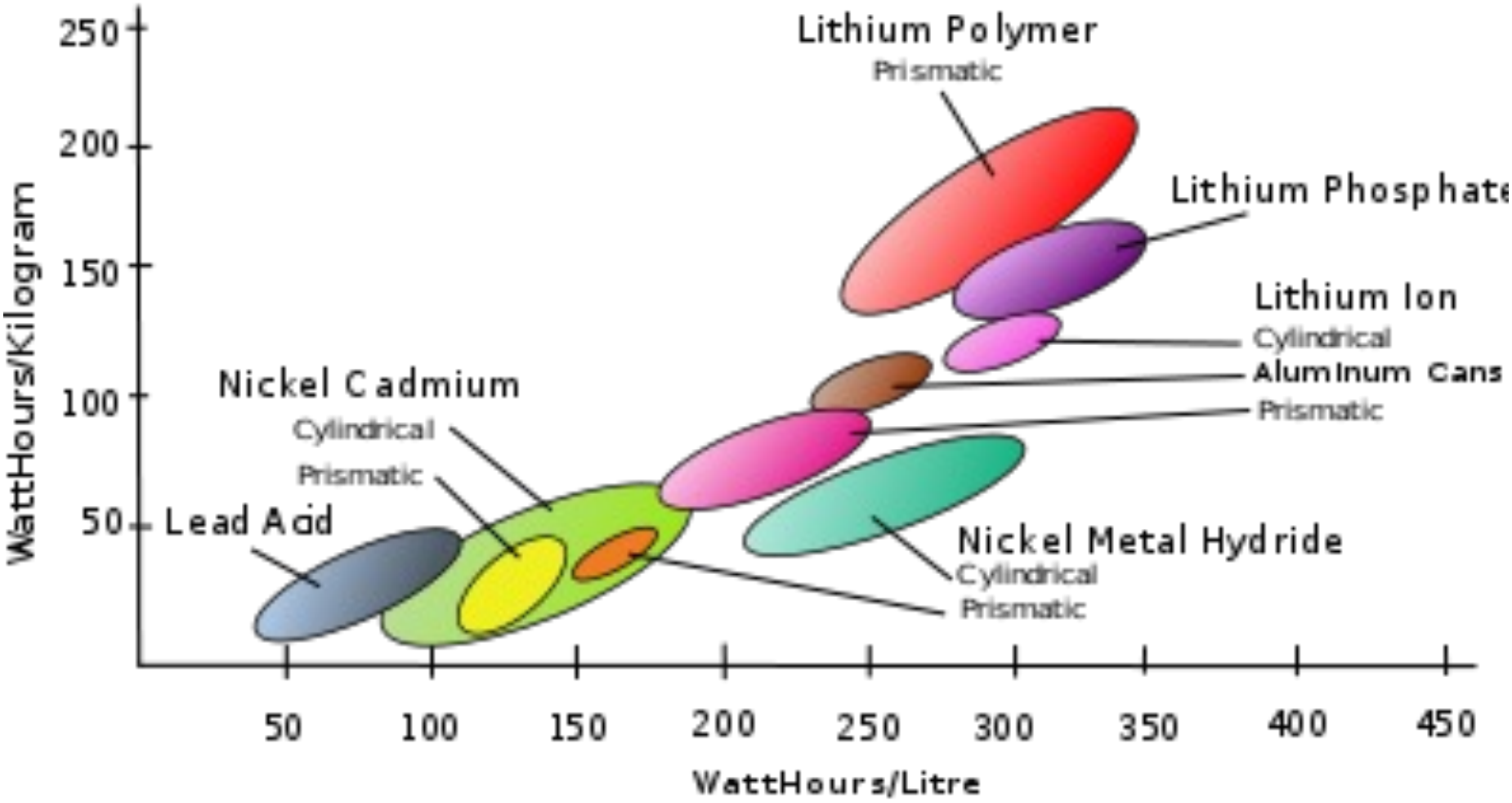
Higher clock frequency -> Higher power dissipation

Reduce supply voltage:

Eg. Change 5 V to 3.3 V supply, saves 56% power

“Sleep when you can”!

Energy Density of Secondary Cells



Source: Wikipedia

Typical Power Consumption

	Email		MP3			Notes		Messaging		
Device	Rcv	Reply	Speaker	Headset	Browse	Text	Audio	Text	Audio	Idle
Laptop	15.16 W	16.25 W	18.02 W	15.99 W	16.55 W	14.20 W	14.65 W	14.40 W	15.50 W	13.975 W
Handheld	1.386 W	1.439 W	2.091 W	1.700 W	1.742 W	1.276 W	1.557 W	1.319 W	-	1.2584 W
Cellphone	539 mW	472 mW	-	-	-	-	-	392 mW	1147 mW	26 mW
Email Pager	92 mW	72 mW	-	-	-	78 mW	-	-	-	13 mW
High-end MP3	-	-	-	2.977 W	-	-	-	-	-	1.884 W
Low-end MP3	-	-	-	327 mW	-	-	-	-	-	143 mW
Voice Recorder	-	-	-	-	-	-	166 mW	-	-	17 mW
variance	*16496%*	*22727%*	*861%*	*4890%*	*950%*	*18252%*	*8825%*	*3673%*	*1351%*	*107500%*

R. N. Mayo and P. Ranganathan. Energy consumption in mobile devices: Why future systems need requirements-aware energy scale-down. In *Proc. of 3rd International Workshop on Power-Aware Computer Systems (PACS)*, pages 26-40, San Diego, CA, Dec. 2003.

Power challenges

- Mostly Battery operated -> Lifetime?
- Recharge-able battery -> Battery recharging issue
- Fully-passive (zero-power) might be the solution
  - Power scavenging or harvesting?
    - Thermal, piezoelectric, pH, solar
  - Inductive or RF powering?
    - RFID, NFC
  - Other powering possibilities?
- Communication range?
  - Wireless communication consumes most power budget!

Power Saving Modes

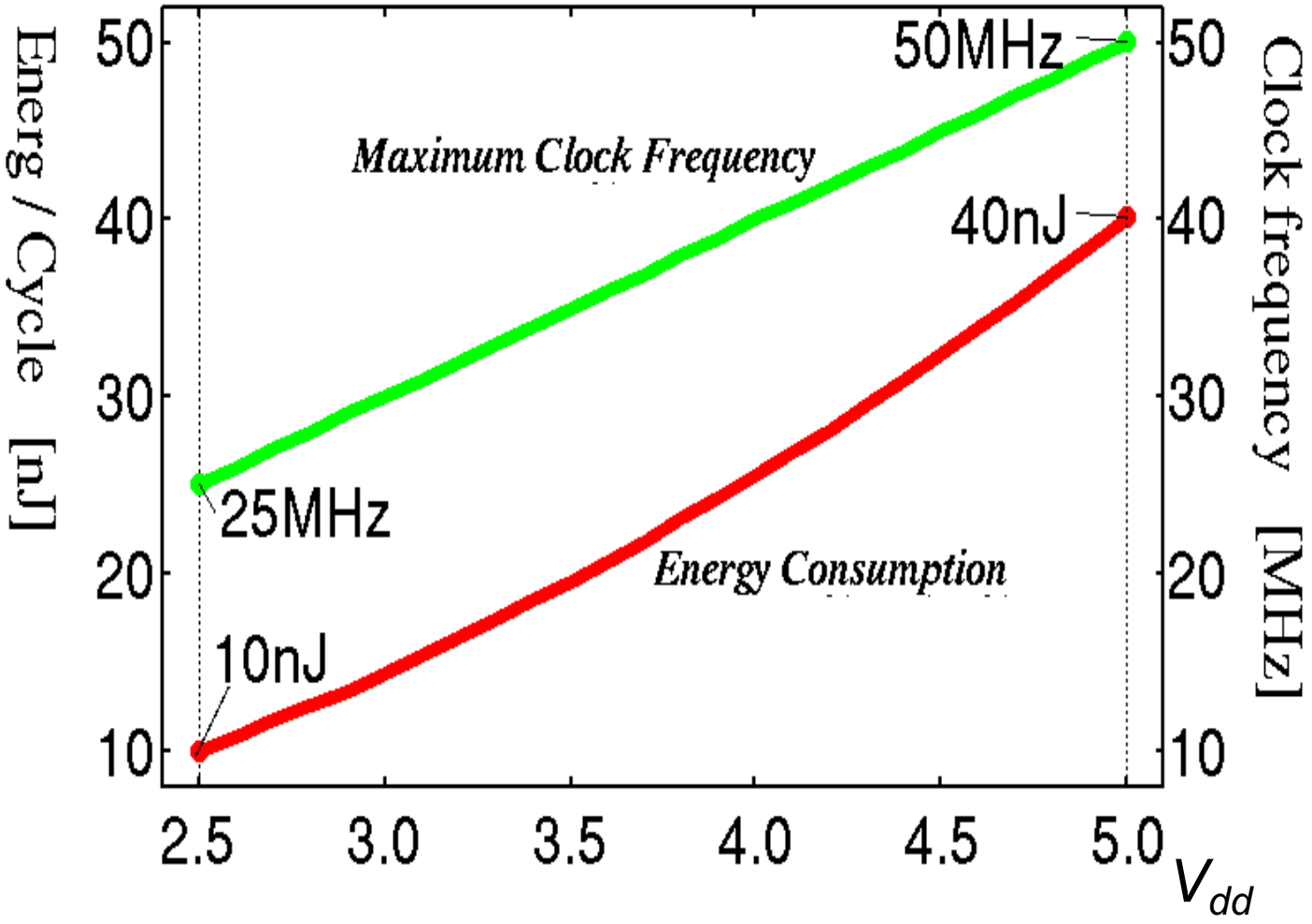
I_{CC}	Supply Current, Active Mode (10)	f = 1MHz, V_{CC} = 2V		0.2	0.4	mA
		f = 4MHz, V_{CC} = 3V		1.4	2.5	mA
		f = 8MHz, V_{CC} = 5V		4.5	8	mA
	Supply Current, Idle Mode	f = 1MHz, V_{CC} = 2V		0.03	0.1	mA
		f = 4MHz, V_{CC} = 3V		0.25	0.6	mA
		f = 8MHz, V_{CC} = 5V		1	2	mA
	Supply Current, Power-down Mode (11)	WDT enabled, V_{CC} = 3V		4	10	µA
		WDT disabled, V_{CC} = 3V		< 0.2	2	µA

	Active Clock Domain				Oscillator	Wake-up Source					
Sleep Mode	clk_{CPU}	clk_{FLASH}	clk_{IO}	clk_{ADC}	Main Clock Source Enabled	INT1, INT0 and Pin Change	TWI Address Match	EEPROM Ready	ADC	WDT	Other I/O
Idle			X	X	X	X	X	X	X	X	X
ADC Noise Reduction				X	X	X(1)	X	X	X	X	
Power-down						X(1)	X			X	

Notes: 1. For INT1 and INT0, only level interrupt

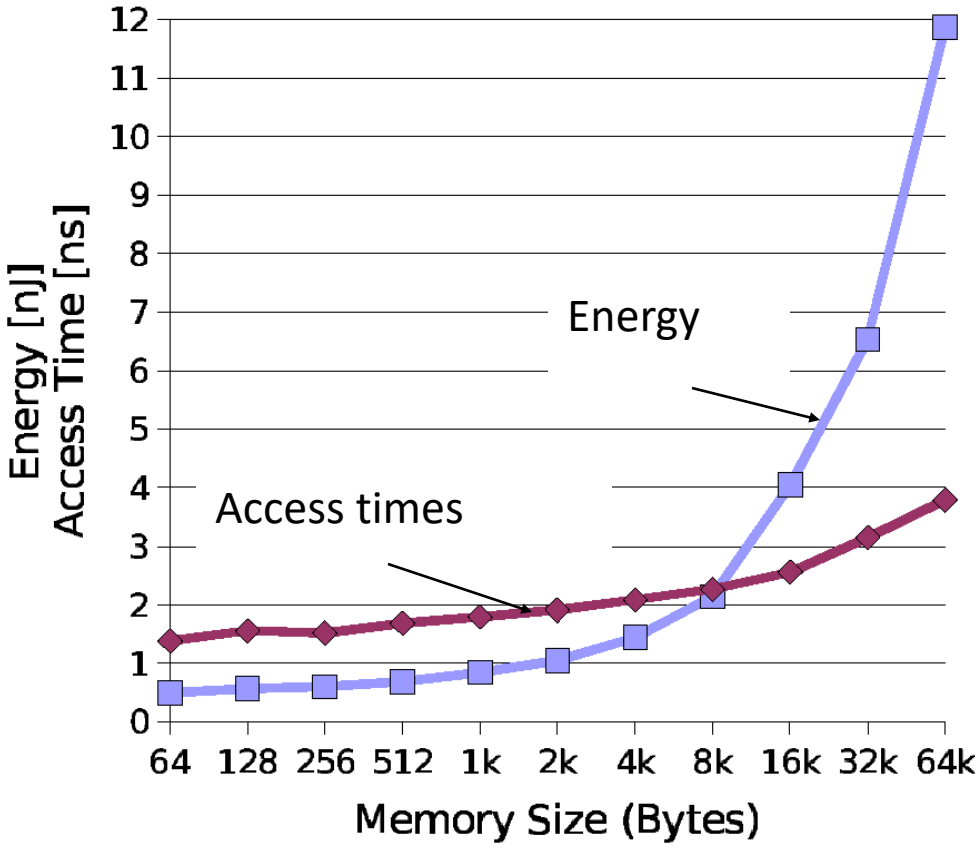
Source: Atmel Attiny88 datasheet

Dynamic Voltage Scaling and Clock Scaling



3 key issues with larger memory

1. (Average) Speed
2. Energy/Power
3. Predictability/WCET



Energy-aware scheduling

Energy-aware scheduling: the order of the instructions can be changes as long as the meaning does not change. Goal: reduction of the number of signal transitions
Popular (can be done as a post-pass optimization with no change to the compiler).

Be aware of Cache!!!

Energy-aware instruction selection: among valid instruction sequences, select those minimizing energy consumption

Exploitation of the memory hierarchy: huge difference between the energy consumption of small and large memories

Energy-aware compilation

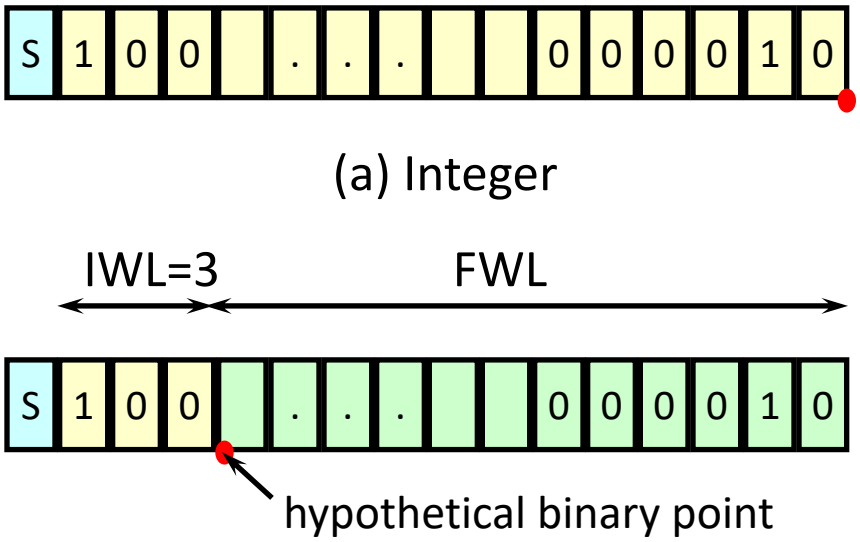
- High-performance if available memory bandwidth fully used; low-energy consumption if memories are at stand-by mode
- Reduced energy if more values are kept in registers
- Operator strength reduction: e.g. replace * by + and <<
- Minimize the bitwidth of loads and stores
- Standard compiler optimizations with energy as a cost function

Fixed-Point vs Floating Point Data

- Floating-Point vs. Fixed-Point

- *exponent*, mantissa
- Floating-Point
  - automatic computation and update of each exponent at run-time
- Fixed-Point
  - implicit exponent
  - determined off-line

- Integer vs. Fixed-Point



Floating-point to fixed point

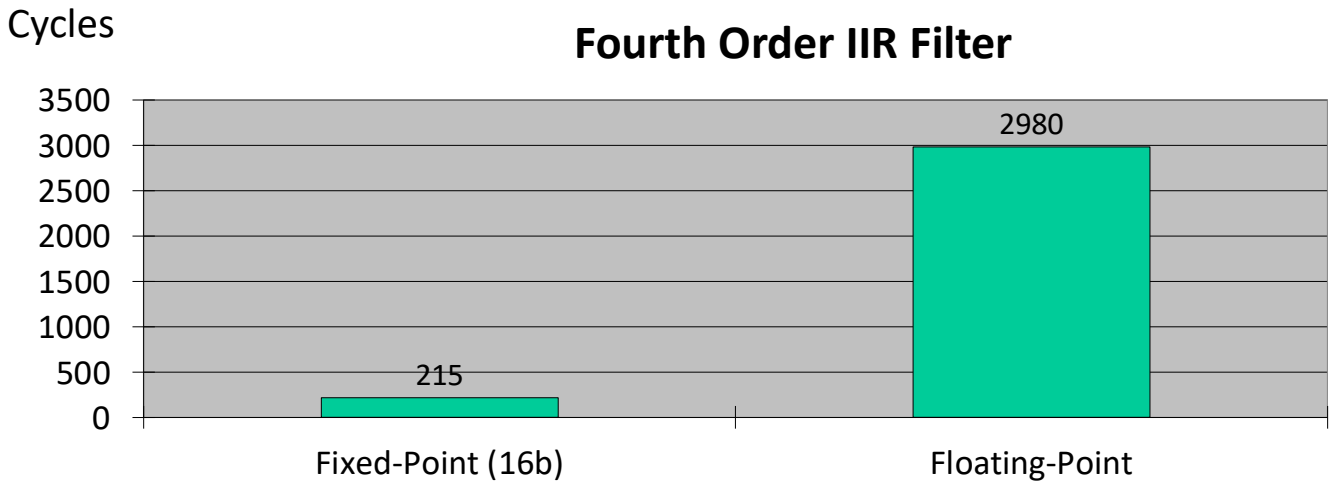
Pros:

- Lower cost
- Faster
- Lower power consumption
- Sufficient SQNR, *if properly scaled*
- Suitable for portable applications

Cons:

- Decreased dynamic range
- Finite word-length effect, *unless properly scaled*
  - Overflow and excessive quantization noise
- Extra programming effort

Performance Comparison: Machine Cycles



Compiler optimization for ES

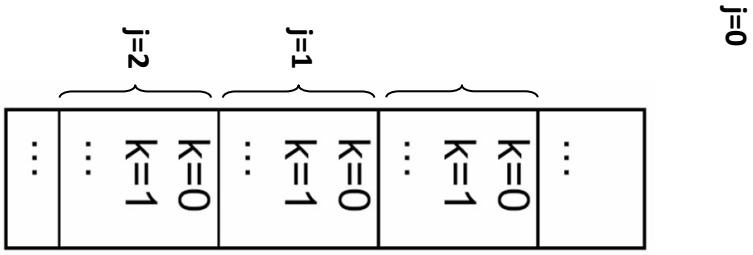
- Many reports about low efficiency of standard compilers
  - Special features of embedded processors have to be exploited.
  - High levels of optimization more important than compilation speed.
  - Compilers can help to reduce the energy consumption.
  - Compilers could help to meet real-time constraints.
- Less legacy problems than for PCs.
  - There is a large variety of instruction sets.
  - Design space exploration for optimized processors makes sense

Higher level optimization: Impact of memory allocation on efficiency

Two loops, assuming
for (k=0; k<=m; k++)
  for (j=0; j<=n; j++))
    p[j][k] = ...

row major order (C):
for (j=0; j<=n; j++)
  for (k=0; k<=m; k++)
p[j][k] = ...

Same behavior for homogeneous memory access, but:



For row major order

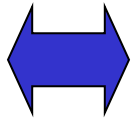
↑ Poor cache behavior Good cache behavior ↑

☞ memory architecture dependent optimization

Best performance if innermost loop corresponds to rightmost array index

Loop Fusion (merging)

```
for(j=0; j<=n; j++)
  p[j]= ... ;
for (j=0; j<=n; j++) ,
  p[j]= p[j] + ...
```



```
for (j=0; j<=n; j++)
  {p[j]= ... ;
  p[j]= p[j] + ...}
```

Loops small enough to allow zero overhead Loops

Better locality for access to p.
Better chances for parallel execution.

Which of the two versions is best?
Architecture-aware compiler should select best version.
Merged loops are usually superior

Loop Unrolling

- **for** (j=0; j<=n; j++)
- p[j]= ... ;

for (j=0; j<=n; j+=2)
{p[j]= ... ; p[j+1]= ...}

factor = 2
Better locality for access to p.
Less branches per execution of the loop. More opportunities for optimizations.
Tradeoff between code size and improvement.
Extreme case: completely unrolled loop (no branch).

Chapter Summary

- Multiple objective analysis
- Pareto Point
- Pareto set
- Pareto front
- Optimization approaches
- Power optimization
- Other optimizations