



Computer Hardware Engineering (IS1200)

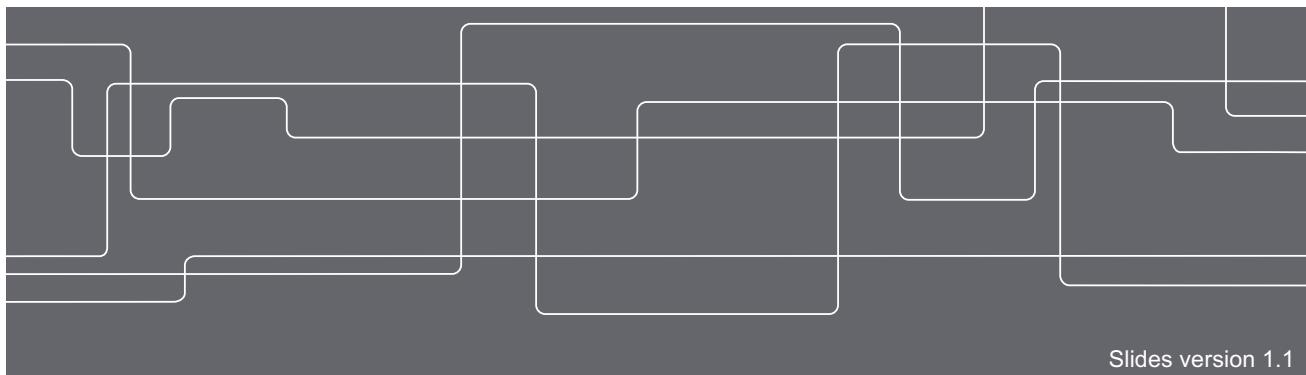
Computer Organization and Components (IS1500)

Spring 2018

Lecture 12: Parallelism, Concurrency, Speedup, and ILP

David Broman

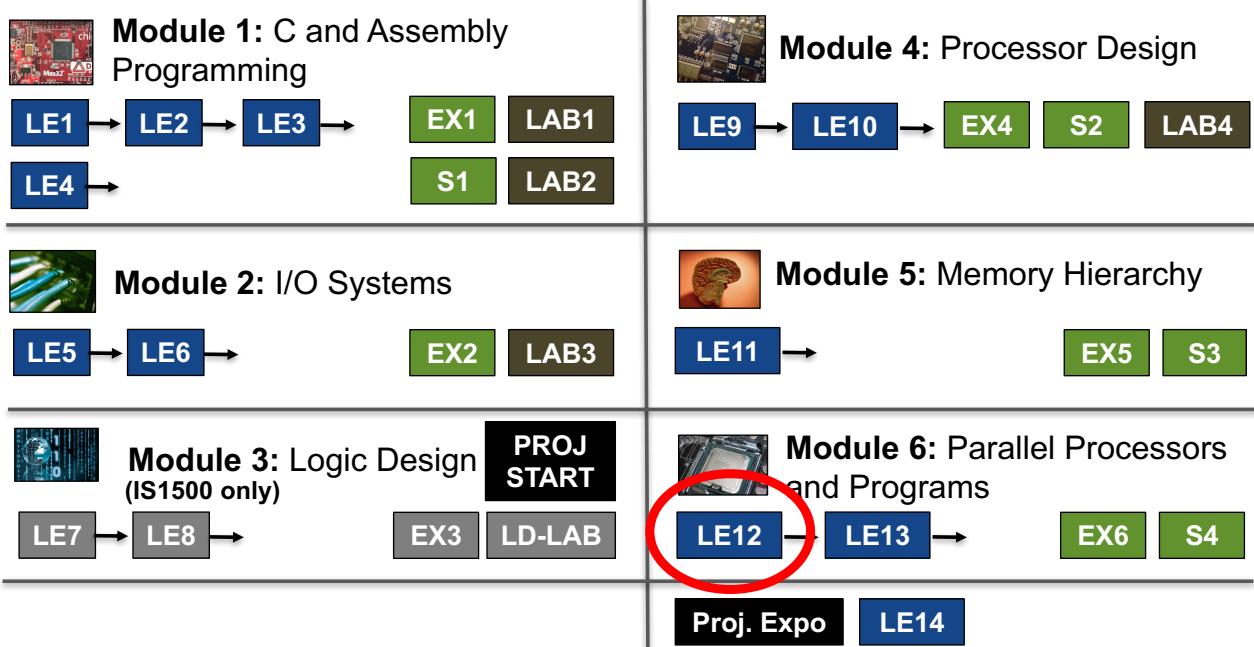
Associate Professor, KTH Royal Institute of Technology



2



Course Structure



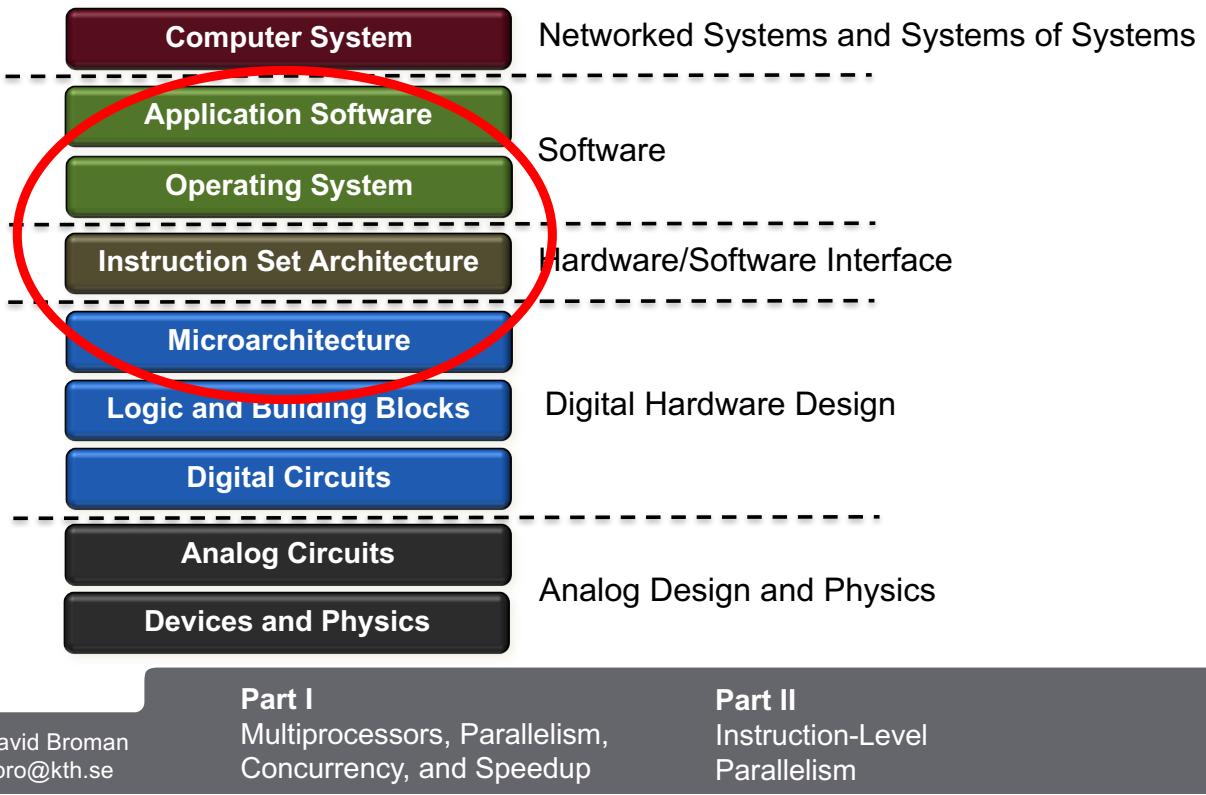
David Broman
dbro@kth.se

Part I
Multiprocessors, Parallelism,
Concurrency, and Speedup

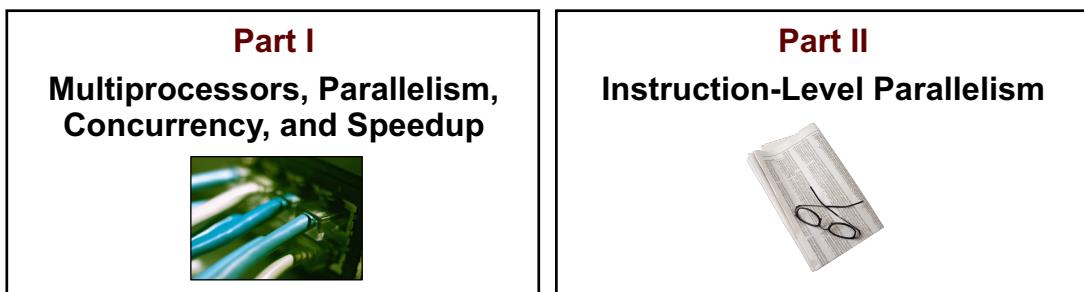
Part II
Instruction-Level
Parallelism



Abstractions in Computer Systems



Agenda



David Broman dbro@kth.se	Part I Multiprocessors, Parallelism, Concurrency, and Speedup	Part II Instruction-Level Parallelism
-----------------------------	--	--



Part I

Multiprocessors, Parallelism, Concurrency, and Speedup



Acknowledgement: The structure and several of the good examples are derived from the book “Computer Organization and Design” (2014) by David A. Patterson and John L. Hennessy

David Broman
dbro@kth.se



Part I
Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II
Instruction-Level
Parallelism



How is this computer revolution possible? (Revisited)

6



Moore's law:

- Integrated circuit resources (transistors) double every 18-24 months.
- By Gordon E. Moore, Intel's co-founder, 1960s.
- Possible because of refined manufacturing processes. E.g., 4th generation Intel Core i7 processors uses 22nm manufacturing.
- Sometimes considered a *self-fulfilling prophecy*. Served as a goal for the semiconductor industry.

David Broman
dbro@kth.se



Part I
Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II
Instruction-Level
Parallelism



Have we reached the limit? (Revisited)

Why?

The Power Wall



http://www.publicdomainpictures.net/view-image.php?image=1281&picture=tegelvagg

During the last decade, the clock rate has increased dramatically.

- 1989: 80486, 25MHz
- 1993: Pentium, 66Mhz
- 1997: Pentium Pro, 200MHz
- 2001: Pentium 4, 2.0 GHz
- 2004: Pentium 4, 3.6 GHz

2016: Intel Core i7, 3.1 GHz (Turbo 3.4Ghz)

Increased clock rate implies increased power

We cannot cool the system enough to increase the clock rate anymore...

“New” trend since 2006: Multicore

- Moore’s law still holds
- More processors on a chip: multicore
- “New” challenge: parallel programming

David Broman
dbro@kth.se



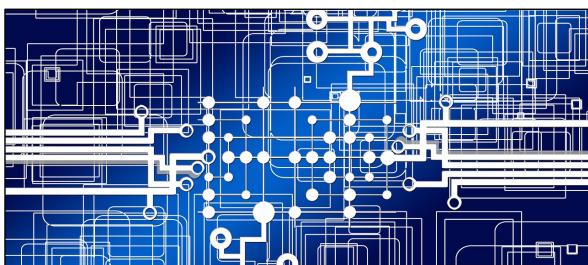
Part I
Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II
Instruction-Level
Parallelism



Moore’s law: we might soon have reached the limit

There is a limit for how small transistors we can manufacture.



By the early 2020s:

“even with superaggressive efforts, we’ll get to the 2–3nanometre limit, where features are just 10 atoms across. Is that a device at all?”

According to Paolo Gargini,
(citation from Waldrop, 2016)

Part of the course literature (especially important for advanced part):
Mitchell Waldrop. **More than More**, *Nature*, Vol 530, 2016. (See the course website)

David Broman
dbro@kth.se



Part I
Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II
Instruction-Level
Parallelism



What is a multiprocessor?

A **multiprocessor** is a computer system with two or more processors.

By contrast, a computer with one processor is called a **uniprocessor**.



by Eric Gaba, CC BY-SA 3.0. No modifications made.

Multicore microprocessors are multiprocessors where all processors (cores) are located on a single integrated circuit.



Photo by Robert Harker

A **cluster** is a set of computers that are connected over a local area network (LAN). May be viewed as one large multiprocessor.

David Broman
dbro@kth.se



Part I
Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II
Instruction-Level
Parallelism



Different Kinds of Computer Systems (Revisited)



Embedded Real-Time Systems



Personal Computers and Personal Mobile Devices



Warehouse Scale Computers

Dependability

Energy

Performance

David Broman
dbro@kth.se

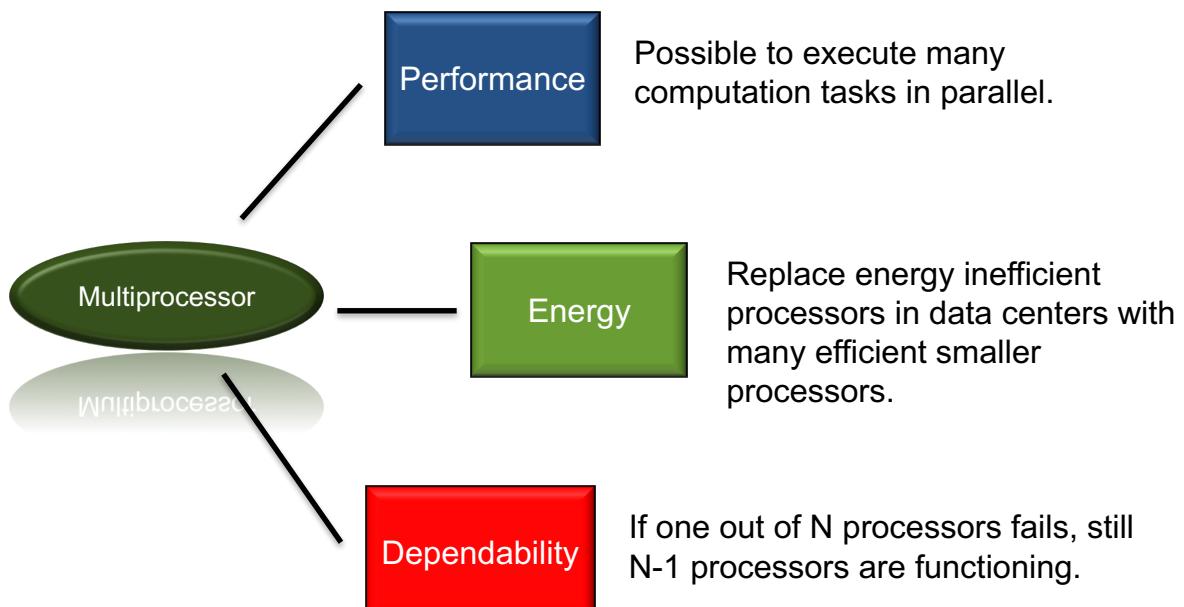


Part I
Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II
Instruction-Level
Parallelism



Why multiprocessors?



David Broman
dbro@kth.se



Part I
Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II
Instruction-Level
Parallelism



Parallelism and Concurrency – what is the difference?

Concurrency is about handling many things at the same time.
Concurrency may be viewed from the **software** viewpoint.

Parallelism is about doing (executing) many things at the same time. Parallelism may be viewed from the **hardware** viewpoint.

Note: As always, everybody does not agree on the definitions of concurrency and parallelism. The matrix is from H&P 2014 and the informal definitions above are similar to what was said in a talk by Rob Pike.

		Software	
		Sequential	Concurrent
Serial		Example: matrix multiplication on a unicore processor.	Example: A Linux OS running on a unicore processor .
	Parallel	Example: matrix multiplication on a multicore processor.	Example: A Linux OS running on a multicore processor.

David Broman
dbro@kth.se



Part I
Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II
Instruction-Level
Parallelism

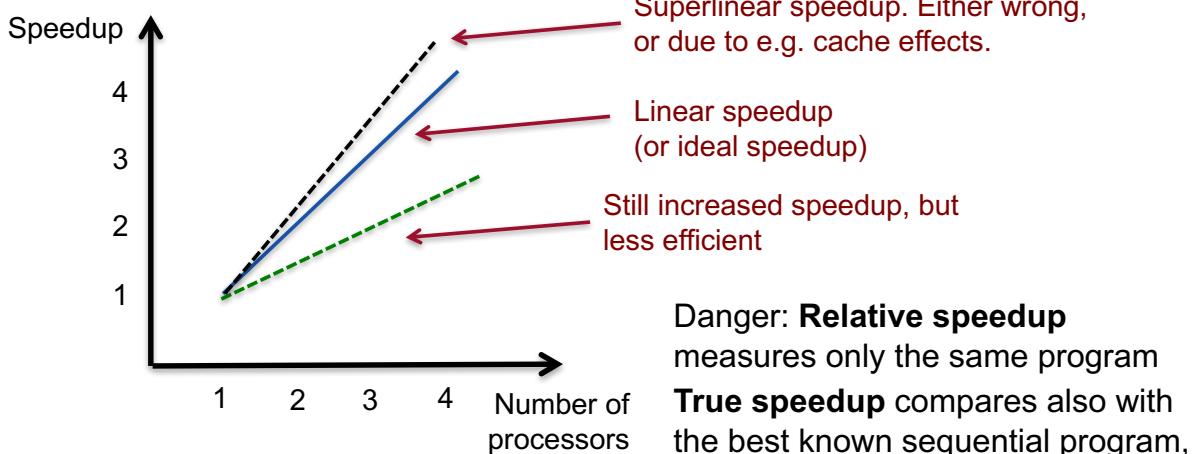


Speedup

How much can we improve the performance using parallelization?

$$\text{Speedup} = \frac{T_{\text{before}}}{T_{\text{after}}}$$

Execution time of one program **before** improvement
Execution time **after** improvement



David Broman
dbro@kth.se



Part I
Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II
Instruction-Level
Parallelism



Amdahl's Law (1/4)

Can we achieve linear speedup?

Divide execution time before improvement into two parts.

Time affected by the improvement of parallelization

Time unaffected of improvement (sequential part)

$$T = T_{\text{affected}} + T_{\text{unaffected}}$$

Execution time after improvement

$$T_{\text{after}} = \frac{T_{\text{affected}}}{N} + T_{\text{unaffected}}$$

Amount of improvement (N times improvement)

$$\text{Speedup} = \frac{T_{\text{before}}}{T_{\text{after}}} = \frac{T_{\text{before}}}{\frac{T_{\text{affected}}}{N} + T_{\text{unaffected}}}$$

This is sometimes referred to as **Amdahl's law**

David Broman
dbro@kth.se



Part I
Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II
Instruction-Level
Parallelism

$$\text{Speedup} = \frac{T_{\text{before}}}{T_{\text{after}}} = \frac{T_{\text{before}}}{\frac{T_{\text{affected}}}{N} + T_{\text{unaffected}}}$$

Exercise: Assume a program consists of an image analysis task, sequentially followed by a statistical computation task. Only the image analysis task can be parallelized. How much do we need to improve the image analysis task to be able to achieve 4 times speedup?

Assume that the program takes 80ms in total and that the image analysis task takes 60ms out of this time.

Solution:

$$4 = 80 / (60 / N + 80 - 60)$$

$$60/N + 20 = 20$$

$$60/N = 0$$

It is impossible to achieve this speedup!

David Broman
dbro@kth.se



Part I
Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II
Instruction-Level
Parallelism



$$\text{Speedup} = \frac{T_{\text{before}}}{T_{\text{after}}} = \frac{T_{\text{before}}}{\frac{T_{\text{affected}}}{N} + T_{\text{unaffected}}}$$

Assume that we perform 10 scalar integer additions, followed by one matrix addition, where matrices are 10x10.

Assume additions take the same amount of time and that we can only parallelize the matrix addition.

Exercise A: What is the speedup with 10 processors?

Exercise B: What is the speedup with 40 processors?

Exercise C: What is the maximal speedup?

Solution A:

$$(10+10*10) / (10*10/10 + 10) = 5.5$$

Solution B:

$$(10+10*10) / (10*10/40 + 10) = 8.8$$

Solution C:

$$(10+10*10) / (10*10/N + 10) = 11 \text{ when } N \rightarrow \infty$$

Alternative solution for C

$$(10+10*10) / (10*10/100 + 10) = 10$$

if we assume that one add instruction cannot be parallelized

David Broman
dbro@kth.se

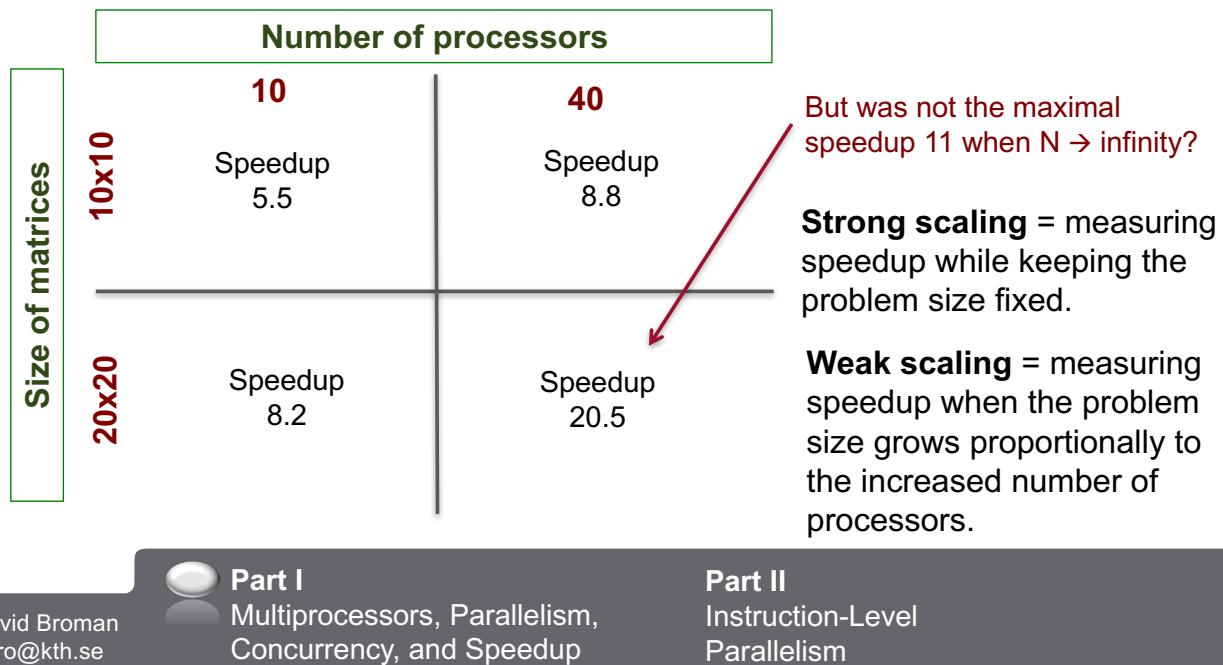


Part I
Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II
Instruction-Level
Parallelism

Amdahl's Law (4/4)

Example continued. What if we change the size of the problem (make the matrices larger)?



Main Classes of Parallelisms



DLP



ДЛР

Data-Level Parallelism (DLP)

Many data items can be processed at the same time.



Example – Sheep shearing

Assume that sheep are data items and the task for the farmer is to do sheep shearing (remove the wool). Data-level parallelism would be the same as using several farm hands to do the shearing.



TLP



ЛГР

Task-Level Parallelism (TLP)

Different tasks of work that can work in independently and in parallel



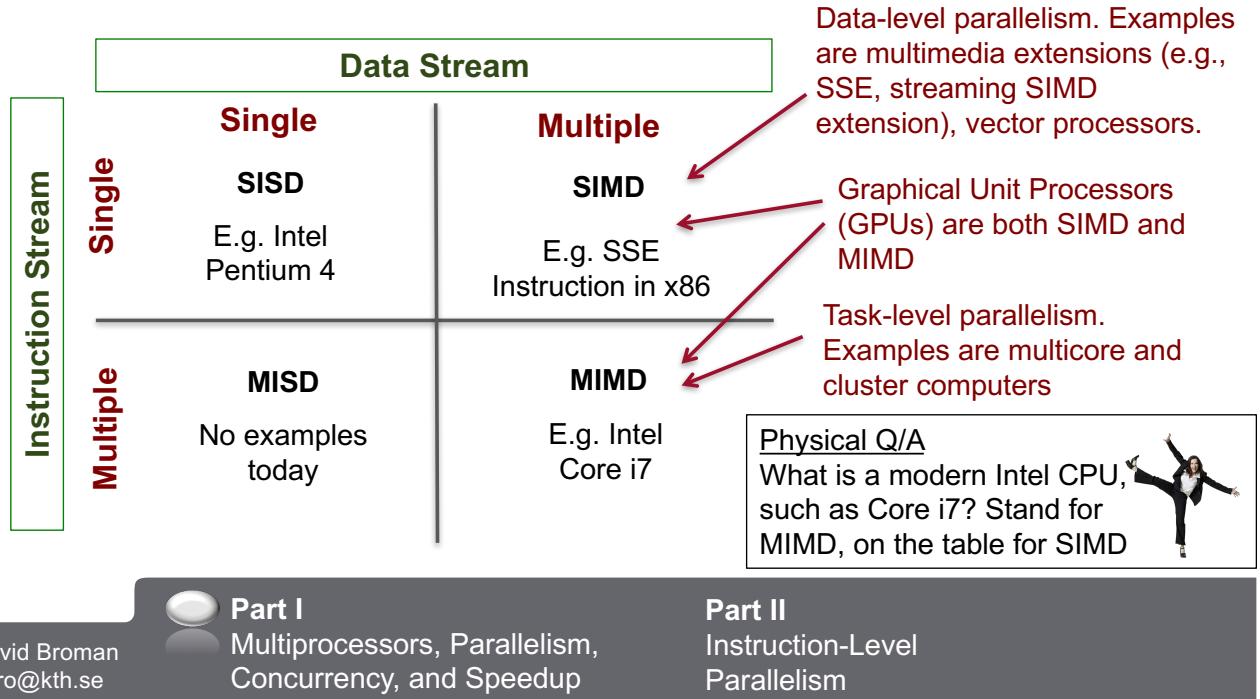
Example – Many tasks at the farm

Assume that there are many different things that can be done on the farm (fix the barn, sheep shearing, feed the pigs etc.) Task-level parallelism would be to let the farm hands do the different tasks in parallel.



SISD, SIMD, and MIMD

An old (from the 1960s) but still very useful classification of processors uses the notion of instruction and data streams.



Part II

Instruction-Level Parallelism



Acknowledgement: The structure and several of the good examples are derived from the book "Computer Organization and Design" (2014) by David A. Patterson and John L. Hennessy



What is Instruction-Level Parallelism?

Instruction-Level Parallelism (ILP) may increase performance without involvement of the programmer. It may be implemented in a SISD, SIMD, and MIMD computer.

Two main approaches:



1. Deep pipelines with more pipeline stages

If the length of all pipeline stages are balanced, we may increase performance by increasing the clock speed.



2. Multiple issue

A technique where multiple instructions are issued in each cycle.

ILP may decrease the CPI to lower than 1, or using the inverse metric *instructions per clock cycle (IPC)* increase it above 1.

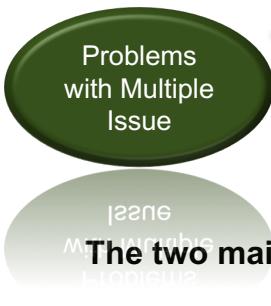
David Broman
dbro@kth.se

Part I
Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II
Instruction-Level
Parallelism



Multiple Issue Approaches



Determining how many and which instructions to issue in each cycle. Problematic since instructions typically depend on each other.

How to deal with data and control hazards

The two main approaches of multiple issue are

1. Static Multiple Issue

Decisions on when and which instructions to issue at each clock cycle is determined by the compiler.

2. Dynamic Multiple Issue

Many of the decisions of issuing instructions are made by the processor, dynamically, during execution.

David Broman
dbro@kth.se

Part I
Multiprocessors, Parallelism,
Concurrency, and Speedup

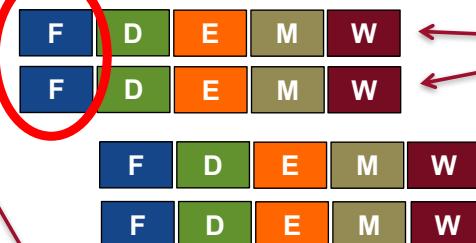
Part II
Instruction-Level
Parallelism

Very Long Instruction Word (VLIW)

processors issue several instructions in each cycle using **issue packages**.

An **issue package** may be seen as one large instruction with multiple operations.

```
add $s0, $s1, $s2
add $t0, $t0, $0
and $t2, $t1, $s0
lw $t0, 0($s0)
```



Each issue package has two **issue slots**.

The compiler may insert **no-ops** to avoid hazards.

How is VLIW affecting the hardware implementation?

David Broman
dbro@kth.se

Part I
Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II
Instruction-Level
Parallelism



Static Multiple Issue (2/3)

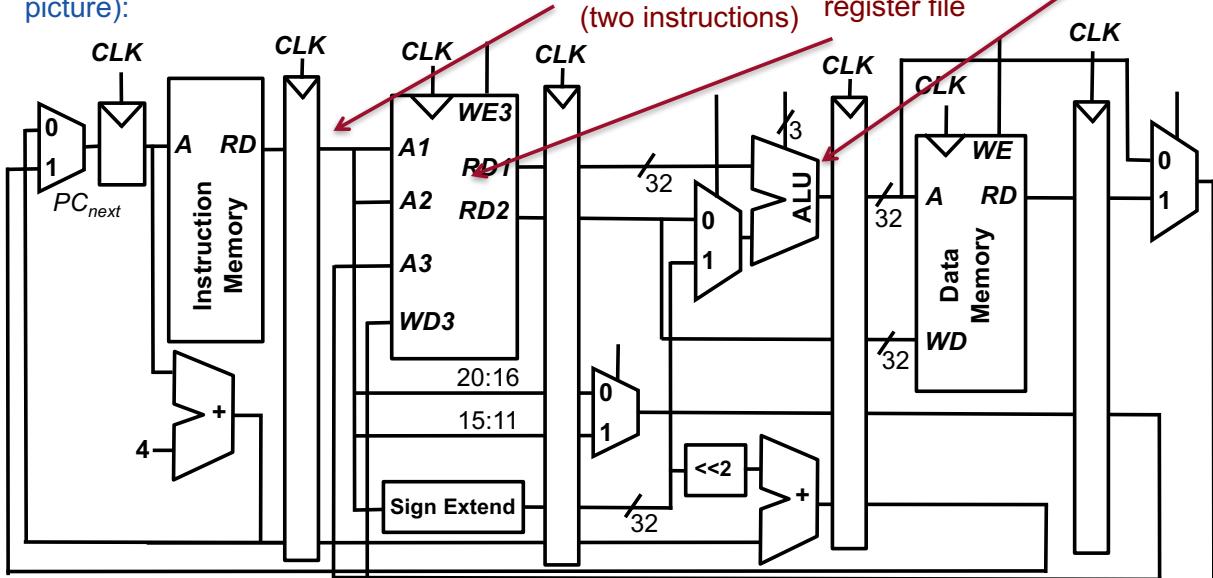
Changing the hardware

To issue two instructions in each cycle, we need the following (not shown in picture):

Fetch and decode 64-bit (two instructions)

Double the number of ports for the register file

Add another ALU



David Broman
dbro@kth.se

Part I
Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II
Instruction-Level
Parallelism

Exercise: Assume we have a VLIW processor that can issue two instructions in the same clock cycle. For the following code, schedule the instructions into issue slots and compute IPC. Assume that no hazards occurs.

Solution. IPC = $(1 + 6*10) / (1 + 4*10) = 1.487\dots$ (max 2)

```

addi $s1, $0, 10
L1: lw    $t0, 0($s2)
ori   $t1, $t0, 7
sw    $t1, 0($s2)
addi $s2, $s2, 4
addi $s1, $s1, -1
bne  $s1, $0, L1

```

	Slot 1	Slot 2	Cycle
	addi \$s1, \$0, 10		1
L1	lw \$t0, 0(\$s2)		2
	addi \$s2, \$s2, 4	addi \$s1, \$s1, -1	3
	ori \$t1, \$t0, 7		4
	bne \$s1, \$0, L1	sw \$t1, -4(\$s2)	5

Empty cells means no-ops

David Broman
dbro@kth.se

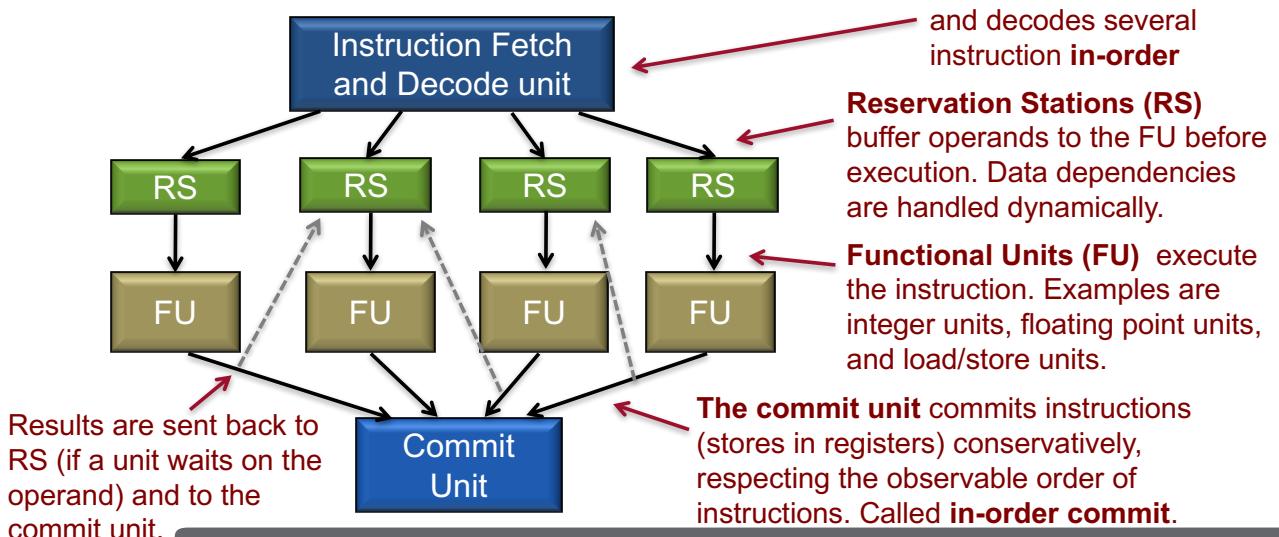
Part I
Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II
Instruction-Level
Parallelism

Dynamic Multiple Issue Processors (1/2)

Superscalar Processors

In many modern processors (e.g., Intel Core i7), instruction issuing is performed dynamically by the processor while executing the program. Such processor is called **superscalar**.



David Broman
dbro@kth.se

Part I
Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II
Instruction-Level
Parallelism

Dynamic Multiple Issue Processors (2/2)

Out-of-Order Execution, RAW, WAR

27

If the superscalar processor can reorder the instruction execution order, it is an **out-of-order execution** processor.

```
lw    $t0, 0($s2)
addi $t1, $t0, 7
```

```
sub  $t0, $t1, $s0
addi $t1, $s0, 10
```

```
addi $r1, $s0, 10
sub  $t0, $t1, $s0
```

Example of a **Read After Write (RAW)** dependency (dependency on \$t0). The superscalar pipeline must make sure that the data is available before read.

Example of a **Write After Read (WAR)** dependency (dependency on \$t1). If the order is flipped due to out-of-order execution, we have a hazard.

WAR dependencies can be resolved using **register renaming**, where the processor writes to a nonarchitectural renaming register (here in the pseudo asm code called \$r1, not accessible to the programmer)

Note that out-of-order processor is not rewriting the code, but keeps track of the renamed registers during execution.

David Broman
dbro@kth.se

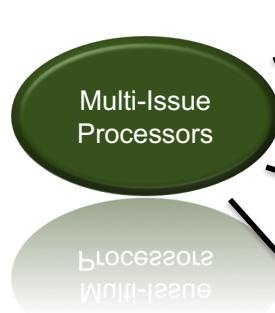
Part I
Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II
Instruction-Level
Parallelism

28



Some observations on Multi-Issue Processors



VLIW processors tend to be more **energy efficient** than superscalar out-of-order processors (less hardware, the compiler does the job)

Superscalar processors with dynamic scheduling can **hide some latencies** that are not statically predictable (e.g., cache misses, dynamic branch predictions).

Although modern processors issues 4 to 6 instructions per clock cycle, few applications results in IPC over 2. The reason is dependencies.

David Broman
dbro@kth.se

Part I
Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II
Instruction-Level
Parallelism



Intel Microprocessors, some examples

Processor	Year	Clock Rate	Pipeline Stages	Issue Width	Cores	Power
Intel 486	1989	25 MHz	5	1	1	5 W
Intel Pentium	1993	66 MHz	5	2	1	10W
Intel Pentium Pro	1997	200 MHz	10	3	1	29 W
Intel Pentium 4 Willamette	2001	2000 MHz	22	3	1	75W
Intel Pentium 4 Prescott	2004	3600 MHz	31	3	1	103W
Intel Core	2006	2930 MHz	14	4	2	75W
Intel Core i5 Nehalem	2010	3300 MHz	14	4	1	87W
Intel Core i5 Ivy Bridge	2012	3400 MHz	14	4	8	77W

Clock rate increase stopped
(the power wall) around 2006

Pipeline stages first increased
and then decreased, but the
number of cores increased
after 2006.

The power consumption
peaked with Pentium 4

Source: Patterson and Hennessey, 2014, page 344.

David Broman
dbro@kth.se

Part I
Multiprocessors, Parallelism,
Concurrency, and Speedup

 **Part II**
Instruction-Level
Parallelism



Bonus Part

Cyber-Physical Systems Design Research Challenges

David Broman @ KTH
(not part of the Examination)



More info, see our research group webpage:
Model-based Computing Systems (MCS), <http://www.kth.se/scs/mcs>

David Broman
dbro@kth.se

Part I
Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II
Instruction-Level
Parallelism



Examples of application areas

Cyber-Physical Systems (CPS)



Automotive
(systems of systems)



Industrial
Automation



Aircraft
(traditional or
autonomous)



Satellites



Medical
Equipment

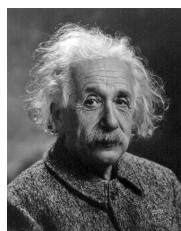
David Broman
dbro@kth.se

Part I
Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II
Instruction-Level
Parallelism



What is our goal?



***“Everything should be made as simple as possible,
but not simpler”***

attributed to Albert Einstein

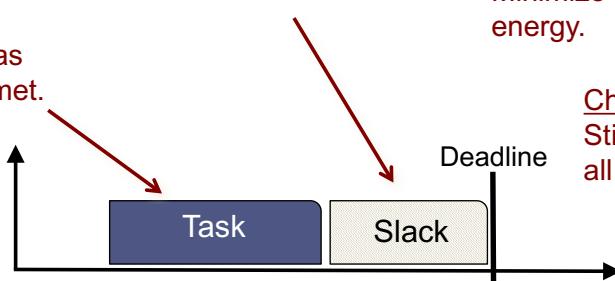
Execution time should be as short as possible, but not shorter

No point in making the execution time shorter, as long as the deadline is met.

Minimize the slack

Objective:
Minimize area, memory,
energy.

Challenge:
Still guarantee to meet all timing constraints.



David Broman
dbro@kth.se

Part I
Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II
Instruction-Level
Parallelism



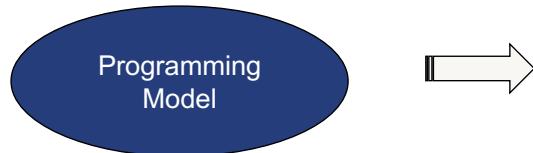
Programming Model and Time



Timing is not part of the software semantics

Correct execution of programs (e.g., in C, C++, C#, Java, Scala, Haskell, OCaml) has nothing to do with how long time things takes to execute.

Traditional Approach



Our Objective



Make time an abstraction within the programming model

Timing Dependent on the Hardware Platform



Timing is independent of the hardware platform (within certain constraints)

David Broman
dbro@kth.se

Part I
Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II
Instruction-Level
Parallelism

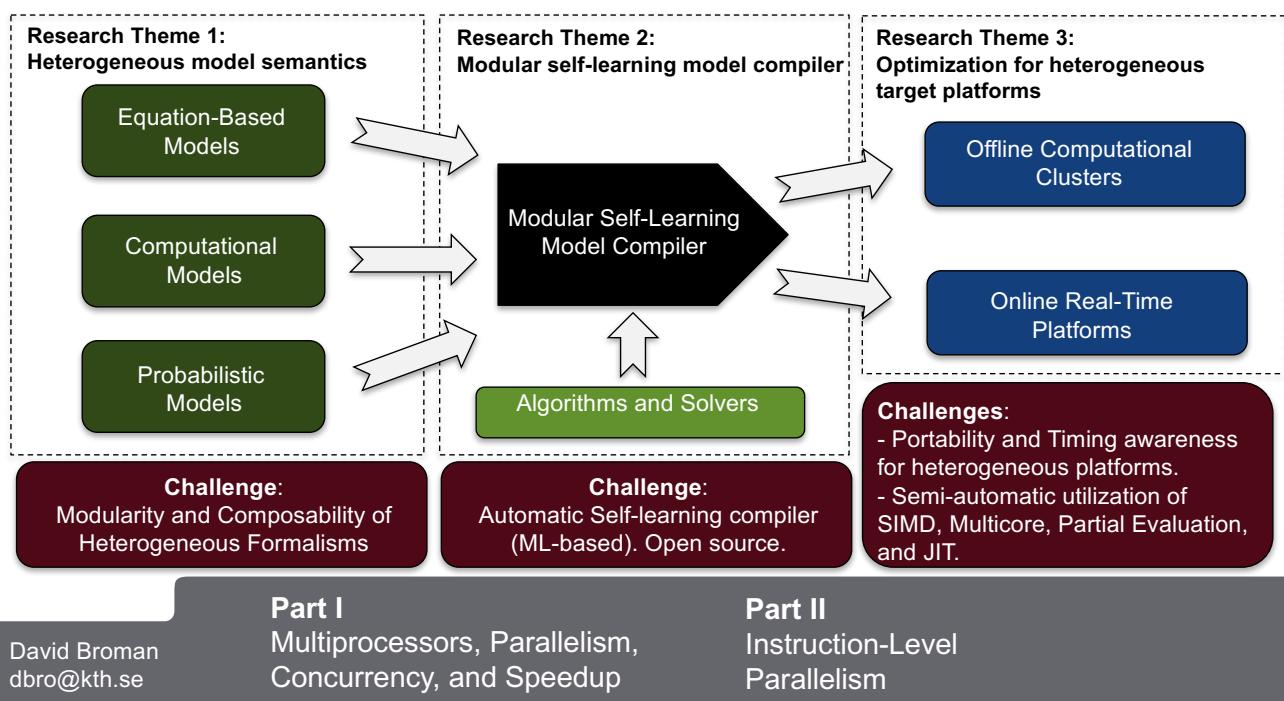


Heterogeneous Model Compilers for Uncertain Environments



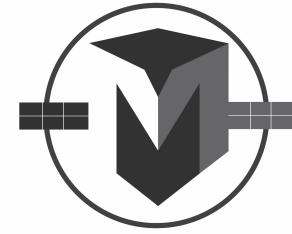
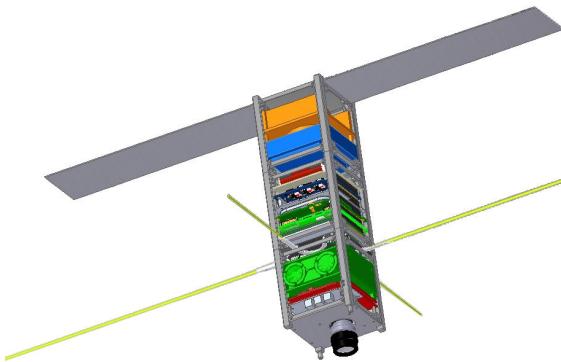
5-year project. Individual grant “Future Research Leaders” from Swedish Foundation for Strategic Research

Financially supported by the Swedish Foundation for Strategic Research.





MIiniature STudent Satellite (MIST)



MIST

- 3-year project, 3U Cubesat (≤ 4 kg).
- Sven Grahn (Project Leader)
Christer Fuglesang (KTH Space Center)
- Built by students at KTH
- 7 scientific experiments on board
- On-board computer (OBC) coordinates experiments and communicates with the ground station on earth.
- Challenges: controlling, testing, and verifying the satellite before launch (attitude, power, communication etc.)

David Broman
dbro@kth.se

Part I
Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II
Instruction-Level
Parallelism



Are you interested to be challenged?

If you are interested in

- Programming language design, or
- Compilers, or
- Machine Learning



You are

- ambitious and interested in learning new things

You want to

- do a real research project as part of your Bachelor's or Master's thesis project

Please send an email to dbro@kth.se, so that we can discuss some ideas.

David Broman
dbro@kth.se

Part I
Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II
Instruction-Level
Parallelism

You know the drill... ...keep calm



David Broman
dbro@kth.se

Part I
Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II
Instruction-Level
Parallelism



Reading Guidelines



Module 6: Parallel Processors and Programs

Lecture 12: Parallelism, Concurrency, Speedup, and ILP

- H&H Chapter 1.8, 3.6, 7.8.3-7.8.5
- P&H5 Chapters 1.7-1.8, 1.10, 4.10, 6.1-6.2
or P&H4 Chapters 1.5-1.6, 1.8, 4.10, 7.1-7.2

Lecture 13: SIMD, MIMD, and Parallel Programming

- H&H Chapter 7.8.6-7.8.9
- P&H5 Chapters 2.11, 3.6, 5.10, 6.3-6.7
or P&H4 Chapters 2.11, 3.6, 5.8, 7.3-7.7

Reading Guidelines

See the course webpage
for more information.

David Broman
dbro@kth.se

Part I
Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II
Instruction-Level
Parallelism



Summary

Some key take away points:

- **Amdahl's law** can be used to estimate maximal speedup when introducing parallelism in parts of a program.
- **Instruction-Level Parallelism (ILP)** has been very important for performance improvements over the years, but improvements have not been as significant lately.



Thanks for listening!

David Broman
dbro@kth.se

Part I
Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II
Instruction-Level
Parallelism