

# Parallel Programming

## Introduction

**Diego Fabregat-Traver** and Prof. Paolo Bientinesi

HPAC, RWTH Aachen  
`fabregat@ices.rwth-aachen.de`

WS15/16



## Acknowledgements

- Prof. Felix Wolf, TU Darmstadt
- Prof. Matthias Müller, ITC, RWTH Aachen

## References

- Computer Organization and Design. *David A. Patterson, John L. Hennessy*. Chapters 1, 4.5 and 4.10.
- Computer Architecture: A Quantitative Approach. *John L. Hennessy, David A. Patterson*. Chapter 1.

# Who has ever heard of ...

---

- ... processors?

# Who has ever heard of ...

---

- ... processors?
- ... frequency/clock (of a processor)?

# Who has ever heard of ...

---

- ... processors?
- ... frequency/clock (of a processor)?
- ... Moore's law?

# Who has ever heard of ...

---

- ... processors?
- ... frequency/clock (of a processor)?
- ... Moore's law?
- ... cache lines? cache associativity?

# Who has ever heard of ...

---

- ... processors?
- ... frequency/clock (of a processor)?
- ... Moore's law?
- ... cache lines? cache associativity?
- ... pipelined and superscalar processors?

# Who has ever heard of ...

---

- ... processors?
- ... frequency/clock (of a processor)?
- ... Moore's law?
- ... cache lines? cache associativity?
- ... pipelined and superscalar processors?
- ... processes and threads?



# Who has ever heard of ...

---

- ... processors?
- ... frequency/clock (of a processor)?
- ... Moore's law?
- ... cache lines? cache associativity?
- ... pipelined and superscalar processors?
- ... processes and threads?
- ... shared and distributed memory?

# Who has ever heard of ...

---

- ... processors?
- ... frequency/clock (of a processor)?
- ... Moore's law?
- ... cache lines? cache associativity?
- ... pipelined and superscalar processors?
- ... processes and threads?
- ... shared and distributed memory?
- ... OpenMP and MPI?

# Outline

---

- 1 Why Parallel Programming?
- 2 Uniprocessor Architecture Review
- 3 Towards the Multi-core Era

## Why Parallel Computing?

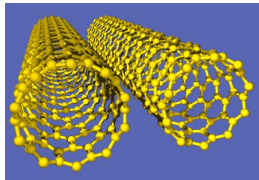
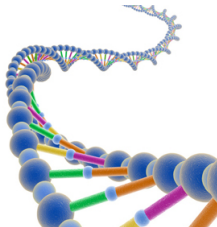
- Problems that cannot be solved fast enough sequentially
  - Real-time constraints
  - Large data sets
  - Accuracy requirements
- Main idea:
  - Decompose large problems into subproblems ...
  - ... that can be solved concurrently.

# Motivation

## Examples

### Computational science

- Genome analysis
- Drug development
- Material science
- Weather forecast
- Climate



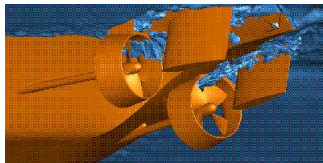
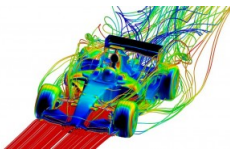
# Motivation

---

## Examples

### Engineering

- Engine design
- Aerodynamics
- Fluid dynamics
- Crash simulations



# Motivation

---

## Examples

### Finance

- Economics
- High-Frequency Trading



## Challenges

- Real-time constraints: 20ns make a huge difference in HFT
- Very large data sets: genome studies routinely process TBs of data

## Computer Simulations

- 3rd pillar of science alongside with theory and experimentation
- Experimentation may be cost prohibitive (e.g., flight testing)
- Experimentation may be impossible (e.g., interaction between space station and spaceship when docking)



- List of the 500 fastest supercomputers in the world
- Twice per year (ISC, June, Germany - SC, November, USA)
- Computers ranked based on the LINPACK benchmark
  - Solution of linear system of equations:  $Ax = b$
  - Result measured in Flop/s<sup>1</sup> (in double precision).
- Established in 1993: 60 GFlop/s
- Latest, June 2015: 33,862,700 GFlop/s

---

<sup>1</sup>Floating Point Operations per Second

# Examples of supercomputers



**Tianhe 2.** Source: top500.org

## Tianhe 2 (Rank #1)

- Site: National SC Center, Guangzhou, China.
- 16,000 compute nodes
  - 2 Intel Ivy Bridge 12-core CPUs
  - 3 Intel Xeon Phis
- 3,120,000 cores
- 1,024,000 GBs
- 33.862,7 TFlop/s (54,902.4 TFlop/s)
- 17,808.00 kW

# Examples of supercomputers

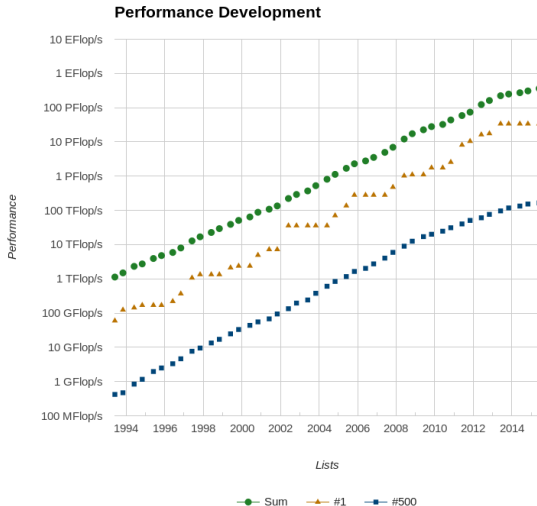


**JuQueen.** Source: fz-juelich.de

## JuQueen (Rank #9)

- Site: Forschungszentrum Jülich (FZJ), Germany.
- 28,672 compute nodes
  - 1 IBM PowerPC A2 16-core CPU
- 458,752 cores
- 458,752 GBs
- 5,008.86 TFlop/s (5,872.03 TFlop/s)
- 2,301.00 kW

# The Top500 List





Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	Tianhe-2	3,120,000	33,862.7	54,902.4	17,808
2	Titan	560,640	17,590.0	27,112.5	8,209
3	Sequoia	1,572,864	17,173.2	20,132.7	7,890
4	K computer	705,024	10,510.0	11,280.4	12,660
5	Mira	786,432	8,586.6	10,066.3	3,945

1,000 KW = 1 MW  $\approx$  1M \$ !!!

Exascale challenges: power wall, memory wall, hardware faults, ...

# So...

---

are parallel computers restricted to supercomputing?

# So...

---

are parallel computers restricted to supercomputing?

Not at all!!



# Parallel Computers are everywhere!

---



Source: [hp.com](http://hp.com)



Source: [indiatimes.com](http://indiatimes.com)



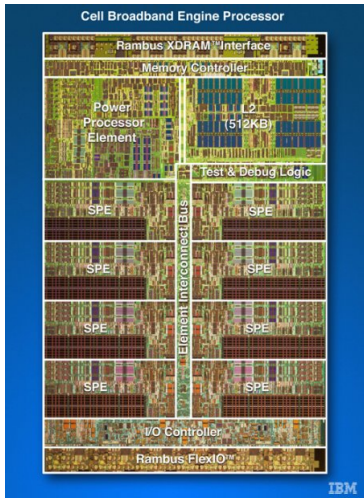
Source: [bq.com](http://bq.com)



Source: [wearabledevices.es](http://wearabledevices.es)

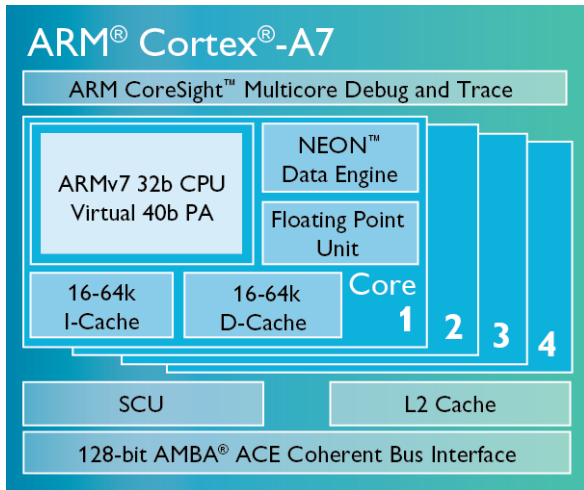
# Parallel Computers are everywhere!

## Play Station 3



# Parallel Computers are everywhere!

My cell phone



Source: arm.com

# Summarizing

---

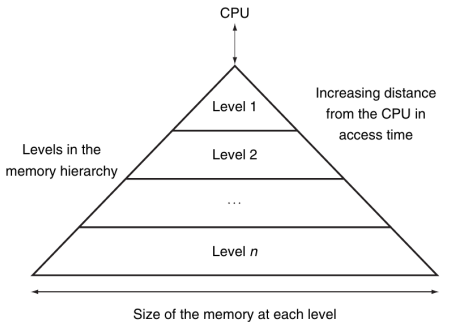
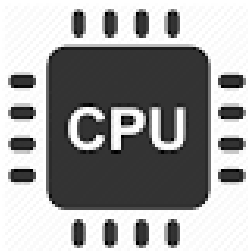
- Parallel programming is critical in science and engineering
- Not only supercomputers, but in every workstation/laptop
- Let's face it:
  - Parallel computers are here to stay
  - The burden is and will be on the programmer
  - So, let's roll up our sleeves and do our best :-)

# Outline

---

- 1 Why Parallel Programming?
- 2 Uniprocessor Architecture Review
- 3 Towards the Multi-core Era

# Quick architecture review



# Clock, cycle, frequency

---

- Clock determines when events take place in the hardware



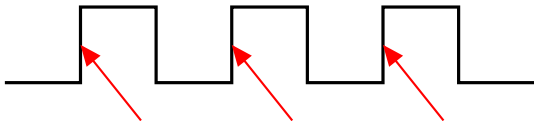
- Frequency (or clock rate): # of cycles per second.

For instance: 2GHz  $\rightarrow 2 \times 10^9$  cycles per second

# Clock, cycle, frequency

---

- Clock determines when events take place in the hardware



- Frequency (or clock rate): # of cycles per second.

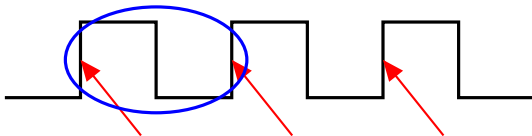
For instance: 2GHz  $\rightarrow 2 \times 10^9$  cycles per second



# Clock, cycle, frequency

---

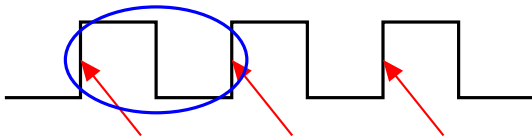
- Clock determines when events take place in the hardware



# Clock, cycle, frequency

---

- Clock determines when events take place in the hardware

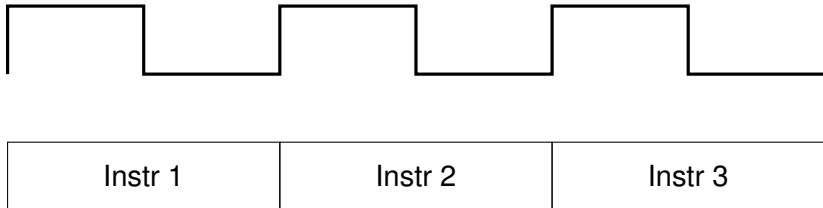


- Frequency (or clock rate): # of cycles per second.

For instance: 2GHz  $\rightarrow 2 \times 10^9$  cycles per second

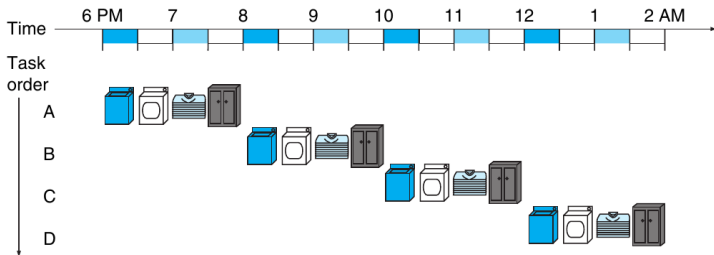
# Purely sequential processor

---



- Latency: 1 cycle
- Throughput: 1 instruction per cycle (IPC)
- Average cycles per instruction (CPI) =  $\frac{1}{IPC}$
- Execution time =  $\#instr * CPI / \text{Frequency}$
- **Inefficient:** long cycles, determined by the slowest instr.

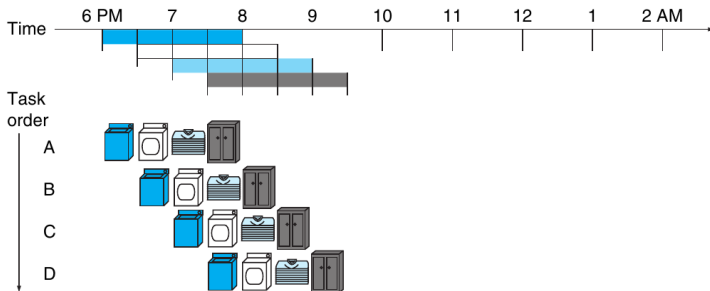
# The laundry analogy



**Source:** *Computer organization and design*. Patterson, Hennessy.

- Latency: 1 load takes 2 hours
- Throughput: 4 loads take 8 hours,  $\frac{1}{2}$  load per hour
- How can we improve the throughput? *Pipelining*

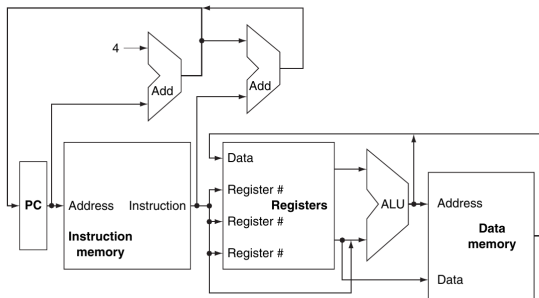
# The laundry analogy



**Source:** *Computer organization and design*. Patterson, Hennessy.

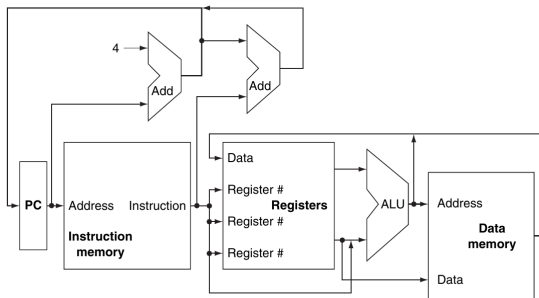
- Latency: still 2 hours
- Throughput:  $(\frac{n}{2+(n-1)*0.5})$  loads per hour ( $\frac{4}{3.5} \approx 1.14$  loads/hour)
- $\lim_{n \rightarrow \infty} \text{Throughput} = 2$  (vs original  $\frac{1}{2}$ )

# Basic processor architecture (MIPS)



**Source:** *Computer organization and design*. Patterson, Hennessy.

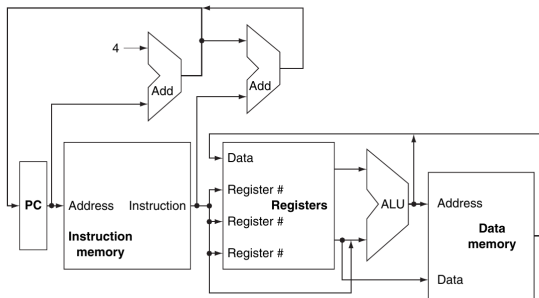
# Basic processor architecture (MIPS)



**Source:** *Computer organization and design*. Patterson, Hennessy.

- **Instruction Fetch (IF):** read instruction from cache

# Basic processor architecture (MIPS)

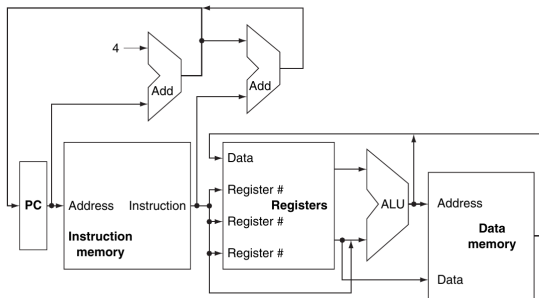


**Source:** *Computer organization and design*. Patterson, Hennessy.

- **Instruction Fetch (IF):** read instruction from cache
- **Instruction Decode (ID):** read register data



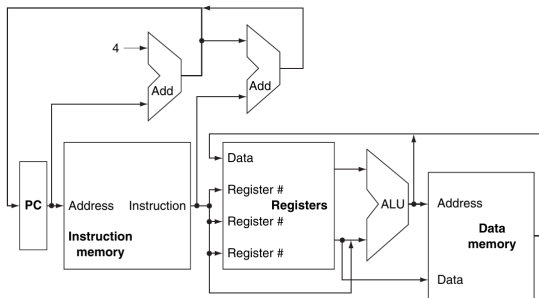
# Basic processor architecture (MIPS)



**Source:** *Computer organization and design.* Patterson, Hennessy.

- **Instruction Fetch (IF):** read instruction from cache
- **Instruction Decode (ID):** read register data
- **Execute (EX):** execute arithmetic/logic operation

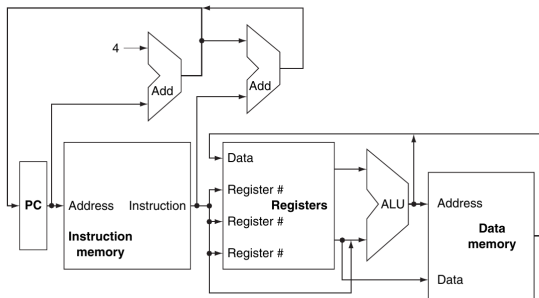
# Basic processor architecture (MIPS)



**Source:** *Computer organization and design*. Patterson, Hennessy.

- **Instruction Fetch (IF):** read instruction from cache
- **Instruction Decode (ID):** read register data
- **Execute (EX):** execute arithmetic/logic operation
- **Memory Access (MEM):** load/store data from/to memory

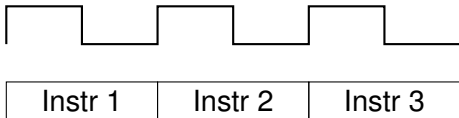
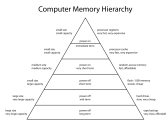
# Basic processor architecture (MIPS)



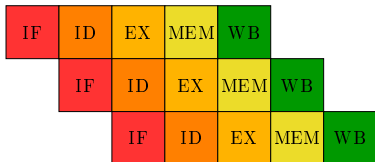
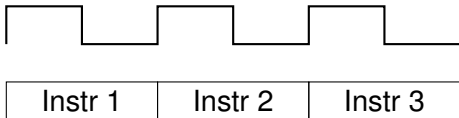
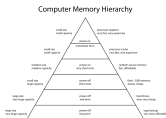
**Source:** *Computer organization and design*. Patterson, Hennessy.

- **Instruction Fetch (IF):** read instruction from cache
- **Instruction Decode (ID):** read register data
- **Execute (EX):** execute arithmetic/logic operation
- **Memory Access (MEM):** load/store data from/to memory
- **Write Back (WB):** write result to register file

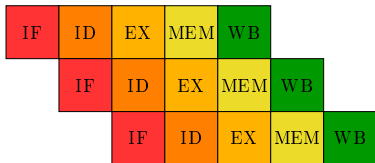
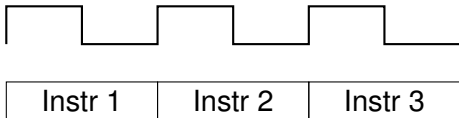
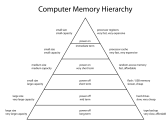
# Pipelined processors



# Pipelined processors



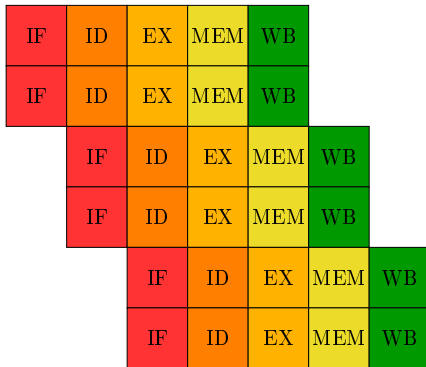
# Pipelined processors



- Each step is known as *stage*
- 5-stage pipeline

# Multiple-issue processors

- Replicate internal components to launch multiple instructions per cycle
- Allows instruction execution rate  $>$  clock rate
- That is, allows to complete the execution of more than one IPC



# Multiple-issue processors

---

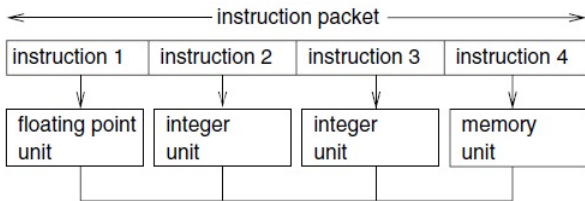
- Adds pressure to the architecture and compiler engineers to keep the processor busy
- Comes in two basic flavors:
  - Static issue
  - Dynamic issue
- The major difference is the division of work between compiler and hardware



# Multiple-issue processors

## Static: Very Large Instruction Word (VLIW)

- Rely on the compiler to schedule and package the instructions
- *Issue packet*: set of instructions to be issued in a given clock cycle
- Think of an issue packet as a large instruction with multiple operations
- Most processors rely on the compiler to take care of data dependencies and such

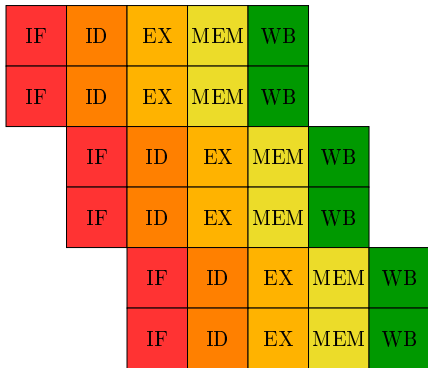


Source: *puyaa.ir*.

# Multiple-issue processors

## Dynamic: Superscalar

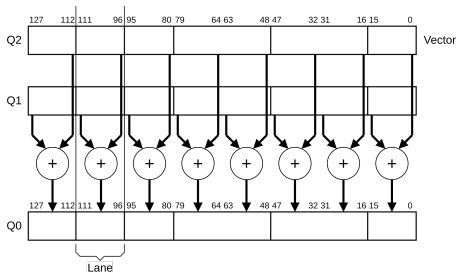
- Processor decides whether to issue zero, one, or more instructions
- Still, compiler helps in scheduling and moving dependencies around
- in-order vs out-of-order



- Width 2
- 2-way superscalar

# Vector units

- Single instruction (single IF, ID)
- Operation applied to multiple data elements
- So called Single-Instruction Multiple-Data (SIMD)



Source: *arm.com*

# Outline

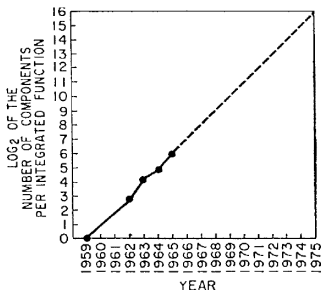
---

- 1 Why Parallel Programming?
- 2 Uniprocessor Architecture Review
- 3 Towards the Multi-core Era

# Towards the multi-core era

## Moore's law

“The number of transistors in a cost-effective circuit doubles every year.” *Cramming more components onto integrated circuits*, 1965.

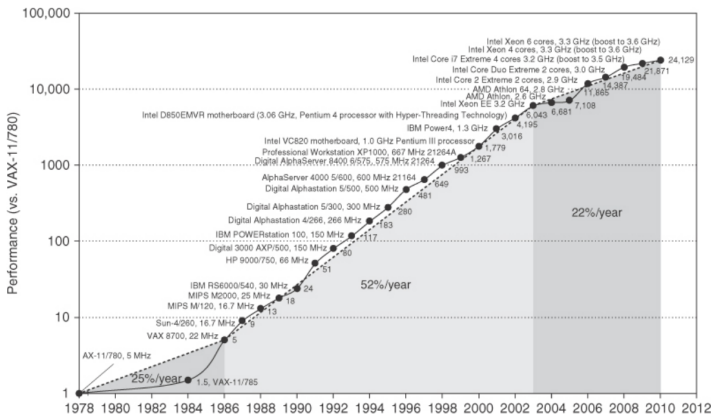


- Updated in 1975:  
... *doubles every two years.*
- Proved to be quite accurate.

Source: Gordon Moore.

# Towards the multi-core era

## Exponential growth in processor performance



**Growth in processor performance since the late 1970s.**

*Computer architecture. Hennessy, Patterson.*

# Towards the multi-core era

---

## Performance growth: Driving forces

### Technology improvements:

- More, faster transistors
- Higher frequency

### Architectural improvements:

- Caches
- Instruction level parallelism
  - Pipelining
  - Multiple instruction issue
  - Dynamic scheduling

# Towards the multi-core era

---

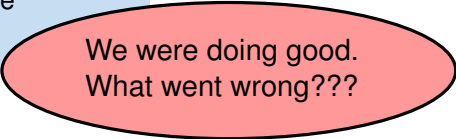
## Performance growth: Driving forces

### Technology improvements:

- More, faster transistors
- Higher frequency

### Architectural improvements:

- Caches
- Instruction level parallelism
  - Pipelining
  - Multiple instruction issue
  - Dynamic scheduling



We were doing good.  
What went wrong???



# Towards the multi-core era

---

## Limitations in ILP

### Trends in multiple-issue processors.

	486	Pentium	Pentium II	Pentium 4	Itanium	Itanium 2	Core2
Year	1989	1993	1998	2001	2002	2004	2006
Width	1	2	3	3	3	6	4

# Towards the multi-core era

## Limitations in ILP

**Trends in multiple-issue processors.**

	486	Pentium	Pentium II	Pentium 4	Itanium	Itanium 2	Core2
Year	1989	1993	1998	2001	2002	2004	2006
Width	1	2	3	3	3	6	4

- High-performance processors:
  - Issue width has stabilized at 4-6
  - Alpha 21464 (8-way) was canceled (2001).
  - Need hardware/compiler scheduling to exploit the width
- Embedded/Low-power processors:
  - Typical width of 2
  - Simpler architectures, no advanced scheduling

# Towards the multi-core era

## Limitations in ILP

Microarchitecture	Pipeline stages
i486	3
P5 (Pentium)	5
P6 (Pentium Pro/II)	14
P6 (Pentium 3)	8
P6 (Pentium M)	10
NetBurst (Northwood)	20
NetBurst (Prescott)	31
Core	12
Nehalem	20
Sandy Bridge	14
Haswell	14

**Table:** Evolution of the pipeline depth for a sample of Intel microarchitectures.

Source: [wikipedia.org](https://en.wikipedia.org/wiki/Intel_Pipeline)

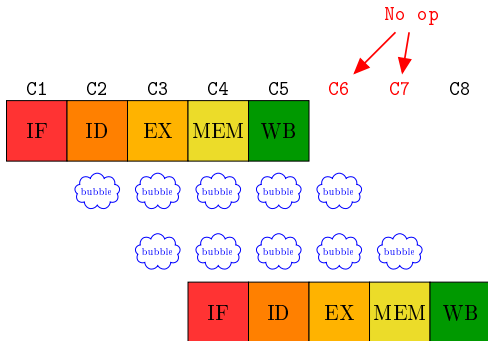
# Towards the multi-core era

## Limitations in ILP

- Data dependencies create bubbles in the pipeline
- Bubbles create delays. Some cycles produce no output.

### Data dependencies

```
load $r0, a  
addi $r1, $r0, 1
```



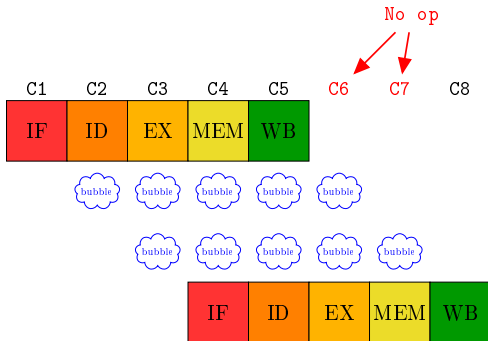
# Towards the multi-core era

## Limitations in ILP

- Branches create bubbles in the pipeline
- Bubbles create delays. Some cycles produce no output.

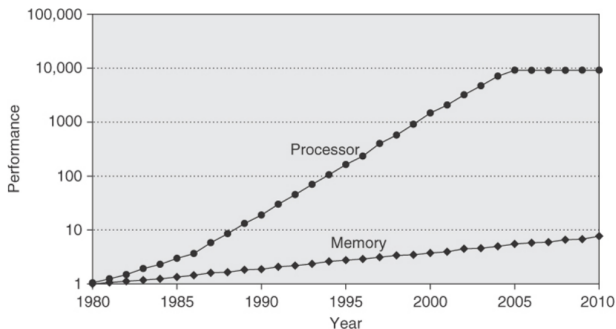
### Control hazard

```
bne $r0, $r1, L2
L1: addi $r2, $r2, 1
...
L2: addi $r2, $r2, -1
```



# Towards the multi-core era

## Divergence CPU-Memory performance

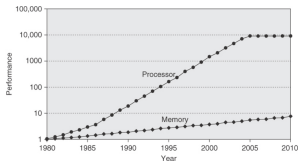


**Gap in performance between CPU and main memory.**

*Computer organization and design. Patterson, Hennessy.*

# Towards the multi-core era

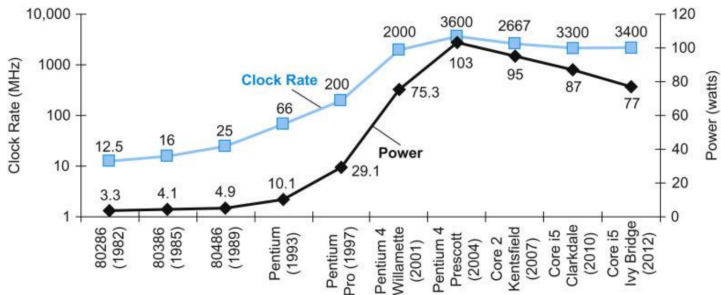
## Divergence CPU-Memory performance



- Performance of both processor and memory grows exponentially
- Gap also grows exponentially
- Processor: 2x every 18 months
- Memory: 2x every 10 years

# Towards the multi-core era

## Frequency stall



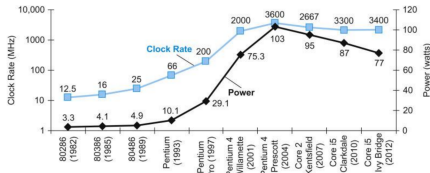
**Clock rate and Power for eight generations of Intel x86 microprocessors.**

*Computer organization and design. Patterson, Hennessy.*



# Towards the multi-core era

## Frequency stall



- Frequency and power consumption are correlated (SC examples)
- Heat is also a problem (P IV Prescott, up to 4GHz, discontinued)
- Trend: simpler pipelines, lower frequency, ..., and multiple cores

- Since 2002/2003 performance of uniprocessors dropped
  - Power dissipation
  - No more instruction-level parallelism to exploit.
- Industry focuses on placing multiple processors on a single die (multi-core architecture)
  - Cores share resources (e.g., caches)
  - Leverages design investment by replicating it

# Multi-cores

---

<b>IBM Proc.</b>	year	# cores
Power 4	2001	2
Power 5	2004	2
Power 6	2007	2
Power 7	2010	4 to 8
Power 8	2013	12

<b>AMD Proc.</b>	year	# cores
Athlon	2005	2
Athlon II	2009	2 to 4
Opteron	2003–	2 to 16

<b>Intel Proc.</b>	year	# cores
Core	2006	2
Core 2	2008	2 to 4
Core i3,i5,i7	2010	2 to 6
Xeon	2006–	2 to 18

# Multi-cores

Available at the ITC Center of RWTH (a sample)

Model	Processor type/ /LSF model name	Sockets/Cores /Threads (total)	Memory Flops/node	Hostname
<b>Bull SMP-XL (BCS)</b> (2 nodes)	Intel Xeon X7550 "Beckton"	4x4 / 128 / 128 2.00 GHz	2 TB 1024 GFlops	linuxbcsc64,65
<b>Bull SMP-D (BCS)</b> (2 nodes)	Intel Xeon X7550	2x4 / 64 / 64 2.00 GHz	256 GB 512 GFlops	cluster cluster-linux
<b>Sun Fire X4170</b> (8 nodes)	Intel Xeon X5570 "Gainestown"	2 / 8 / 16 2.93 GHz	36 GB 93.76 GFlops	linuxnc001..008
<b>Sun Blade X6275</b> (192 nodes)	Intel Xeon X5570 "Gainestown"	2 / 8 / 16 2.93 GHz	24 GB 93.76 GFlops	linuxnc009..200
<b>Sun Fire X4450</b> (9 nodes)	Intel Xeon 7460 "Dunnington"	4 / 24 / - 2.66 GHz	128-256 GB 255.4 GFlops	linuxdc01..09

## New version of Moore's law

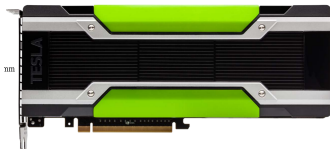
- The number of cores will double every two years
- We will see processors with 100s and 1000s of cores

## Integration and Heterogeneity

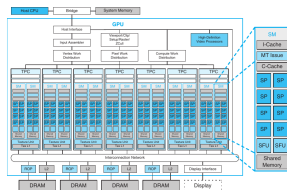
- Cores for specific functions (DSP, Cryptography, ...)
- Integrated on die

# Many-core co-processors

## GPGPUs



Source: nvidia.com



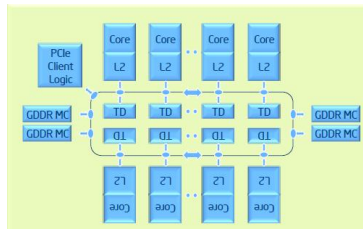
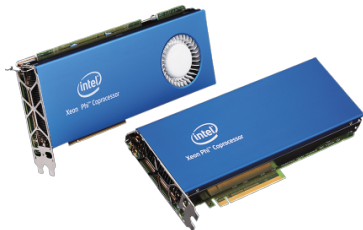
*Computer organization and design.*  
Patterson, Hennessy.

## Tesla K80

- 2x Kepler GK210
- Combined 1.87 TFlop/s peak (double precision)
- 26 Streaming multiprocessors (“cores”)
- 4992 CUDA cores (“FPU”)

# Many-core co-processors

## Intel Xeon Phi



Source: intel.com

- $\approx 1$  TFlop/s peak (double precision)
- 61 cores

# Summary

---

- Why HPC and PP are important
  - Solve large problems fast
  - Simulations in science and engineering enable new research where experiments are cost prohibitive or not possible
- The technology trends
  - Limit power consumption
  - Decrease frequency
  - Increase number of processors/cores
- Free lunch is over
  - No more significant improvements in uni-processor performance
  - Further performance gains via explicit parallelism
  - Will require at very least 100-way concurrency
  - Heterogeneous architectures (Tianhe2)