# Contents

1 Introduction 1

2 Memory Hierarchy 9

3 Microarchitecture and the Pipeline 10

4 Security from the Hardware/Systems Perspective 10

5 Multicore and Heterogeneous Systems 10

6 Power and Energy 10

7 Section 10

8 Appendix 10

# 1 Introduction

## Readings

- Chapter 1, 2, 5, Appendix B

## Priorities of Computer Architecture Designs

- Performance
- Cost/Price
- Energy and Pwer
- Functional requirements
- Dependability
- Trends of technology and computer-use (market)
- Computer Architecture is now more than just the instruction set design.

## Classes of Computers

- Internet of Things (IoT)/Embedded Computers
- Personal Mobile Devices (PMD)
- Desktops
- Servers

- Warehouse-Scale Computers (WSC) / Clusters

## Classes of Parallelism and Parallel Architecture

Two kinds of parallelism in applications:

1. **Data-level parallelism** (DLP): many data items can be operated on at the same time.
2. **Task-level parallelism** (TLP): tasks of work are created that can operate independently and largely in parallel.

How computer hardware exploits parallelism:

1. *Instruction-level parallelism* (ILP): exploits DLP using speculative execution and pipelining.
2. *Vector architectures, graphic processor units (GPUs) and multimedia instruction sets*: exploits DLP by applying a single instruction to a collection of data in parallel.
3. *Thread-level parallelism*: exploits DLP or TLP in tightly coupled hardware models that allow interaction between parallel threads.
4. *Request-level parallelism* (RLP): exploits parallelism between largely decoupled tasks specifiec by programmer or OS.

Four class categories of parallel computers:

1. **Single instruction stream, single data stream** (SISD): uniprocessors, can exploit ILP (ex. superscalars and speculative execution).
2. **Single instruction stream, multiple data streams** (SIMD): one instruction executed by multiple processors using different data streams. Exploits DLP (apply same operations to multiple items of data in parallel, each processor has its own memory but there is a single instruction memory and control processor). Ex. vector architectures, instruction sets with multimedia extensions, and GPUs.
3. **Multiple instruction streams, single data stream** (MISD): theres no commercial multiprocessor that fits this.
4. **Multiple instruction streas, multiple data streams** (MIMD): each processor fetches its own instruction and operates on its own data. Exploits TLP. More general and flexibl but more expensive than SIMD (i.e. MIMD DLP exploits have more costly overhead than DLP exploits in SIMD).

Many parallel computers are hybrids of SISD, SIMD, and MIMD.

# 7 Dimensions of Instruction Set Architecture (ISA)

1. *Class of ISA*: almost all of today's ISAs are general-purpose register architectures. Two popular ones are register-memory ISAs and load-store ISAs. All ISAs since 1985 are

load-store.

2. *Memory Addressing*: byte addressing to access memory operands. Objects don't have to be aligned but alignment tends to make accessing faster.
3. *Addressing Modes*: how to specify address of a memory object. Ex. Register, Immediate, Displacement (ex. PC-ralative address).
4. *Types and Sizes of Operands*
5. *Operations*: op categores are generally: data transfer, arithmetic logical, control, and floating point.
6. *Control-flow Instructions*: support for conditional branches, unconditional jumps, procedure calls, and returns. Notes, depending on ISA: Branching may involve testing value in register or looking at a flag set as side effect of arithmetic/logical operation, and return address may be placed in a register or it may be placed on a stack.
7. *Encoding an ISA*: fixed length (ex. ARMv8 and RISC-V) vs. variable length (ex. 80x86 and other ARMv8 and RISC-V extensions) encoding of an ISA. Programs compiled for variable length ISA can take up less space. This can be impacted by number of registers and number of addressing modes in the ISA.

*

ectionRISC-V Benefits

- Large set of registers
- Easy-to-pipeline instructions
- Lean set of operations
- open-source
- Adopted by many large companies

## Implementation of Computer

1. **Organizaion/Microarchitecture**: computer design (memory system, memory interconnect, internal processor/CPU design (this is where arithmetic, logic, branching, and data transfer are implemented)). Example: two processors with the same ISA can differ in their microarchitecture such as such as in their pipeline and cache design.
    - In multiprocessor microprocessor terms, core means processor, and multicore means multiprocessor microprocessor. Since most chips are multicores, the term CPU is fading.
2. **Hardware**: the specifics of the computer, like the detailed logic of the design and the packaging technology of the computer. Two computers with the same ISAs and

organization may differ in the hardware, such as having different clock rates or different memory systems.

The term *architecture* refers to ISA, microarchitecture, and hardware.

## Technology Trends

Potential 5 rapid changes in implementation technology that affects computer design:

1. **Integrated Circuit Logic Technology**: transistors
2. **Semiconductor DRAM**: main memory
3. **Semiconductor Flash**: nonvolatile memory, electrically erasable programmable read-only memory.
4. **Magnetic Disk technology**: disk and disk density
5. **Network technology**: network performance depends on performance of switches and transmission system.

Performance Trends:

- **Bandwidth/Throughput**: total work done in given time.
- **Latency/Response Time**: time between start and completion of an event.

Bandwidth usually grows by at least square of the improvement in latency.

Scaling of Transistor Performance and Wires

- *feature size* is minimum size of transistor or wire in the x or y dimension.
- density of transistors on an area of silicon increases quadratically as feature size decreases linearly.
- transistor performance in complex because shrinking transistor size requires reducing operating voltage to maintain correct operation and reliability of transistors.
- Wires in an integrated ciruit do not generally improve performance as feature size decrease (resistance and capacitance worsen when feature size shrink (wires shorten), and they are affected by many other factors. Power dissipation limit). Wire delays scale poorly.

## Power and Energy Trends in Integrated Circuits

Energy is the biggest challenge across all computer types for computer designers.

- Power must be brought in and distributed around the chip.
- Power is dissipated as heat must be removed.

Performance, Power, Energy (System Designs)

1. Max power required by processor (upper limit). Drawing more power than what the power-supply system can provide can lead to voltage drop, which causes device malfunction. Slowing down processor to regulate voltage decreases performance.
2. Sustained power consumption / thermal design power (TDP) is the cooling requirements. Device failure and permanent damage can result from exceeding the maximum junction temperature of the processor. As thermal temp approaches junction temp limit, circuitry can lower clock rate to reduce power. If this fails, the chip powers down.
3. Energy and energy efficiency: power is energy per unit time (1 W = 1 J/s). Energy is better than power as a metric because energy is the work and time of a task. Compare energy consuption of two processors for a given task to measure efficiency. Power is useful for measuing constraints on power.

Energy and Power in $\mu$processors

- For a fixed task, slowing clock rate reduces power but not energy
- Dynamic power and energy can be reduced by lowering voltage
- Power consumption and energy grew when transistor density grew, because the transistor switching and frequency with which they changed dominated the decrease in capacitive load and voltage.
- clock frequency growth have slowed since we can't reduce voltage or increase power per chip.
- Distributing power, removing heat, and preventing hot spots are challenges. Before, it was the area of silicon, now it is the energy.
- Addressing energy efficiency despite flattened clock rates and constant supply voltages:
    1. turn off clocks of idle or inactive modules.
    2. Dynamic voltage-frequency scaling (DVFS): offer lower clock frequencies and voltages to operate and use lower power and energy during periods of low activity.
    3. design for the typical/common case.
    4. overclocking: let chip decide when it is safe to run at higher clock rates. temporary overclocks and let safety mechanisms handle heating.
- static power is also an issue now because leakage current flows even when a transistor is off.
- Measurements of task/jooule or performance/watt are important and affects approaches to parallelism.

Energy Limits and Computer Architecture

- New design principles focus on energy per task.

- Domain-specific processors can save energy. While they have a limited set of tasks, they do them faster and more energy efficiently than general-purpose processors.
- We may see a trend in chips with hybrid general-purpose cores and these specialty cores.

## Cost Trends

The goal is to generally use technology improvements to lower cost and increase performance.

- Things like time (lifetime of a product in existence), volume (number and availability of units), and commodity (market competition) lowers cost, either as they increase or over time.
- Integrated circuits factor greatly into a computer's cost.
-

## Dependability

- With how small feature sizes are now, we are facing increasing challenges in the reliabilty of the integrated circuits to not fail.
- Redundancy usualy copes with failure, either with time or resources.

## Quantify Performance

Equations:

| | |
|---|---|
| Performance of process X: | $\text{Performance}_X = \frac{1}{\text{Execution time}_X}$ |
| "X is $n$ times as fast as Y": | $\frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{\text{Execution time}_Y}{\text{Execution time}_X} = n$ |
| CPU time for program (s) | = # CPU clock cycles × clock cycle time<br>= # CPU clock cycles / clock rate |
| # CPU clock cycles for program (cycles) | = # Instructions × CPI |
| Performance Equation: | CPU time (s) = #Instr. × CPI × clock cycle time<br>CPU time (s) = #Instr. × CPI × $\frac{1}{\text{clock rate}}$ |
| Power<br>Relative Power: | $P = C \times V^2 \times F$<br>$\frac{P_{\text{new}}}{P_{\text{old}}}$ |
| Speedup | $= \frac{\text{Exec time}_{\text{no enhancement}}}{\text{Exec time}_{\text{w/enhancement}}}$ |
| Overall speedup | $1/\left((1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}\right)$ |
| $\text{Exec time}_{\text{new}}$ | = $\text{Exec time}_{\text{old}}$ × Overall speedup |

- benchmarking simple programs and trying to correlate results to those of real applications is a pitfall. This is not a good way to benchmark processor performance. i.e. running kernal, toy programs, or synthetic benchmarks. Although synthetic benchmark is still used widely in embedded processors, despite its unreliability.
- Benchmark suites are used (SPEC)
- Reporting Performance Requirements:
  - Reproducibility
  - extensive description of computer and compiler flags
  - publication of both baseline and optimized results
  - HW, SW, and baseline tuning parameter descriptions

- actual performance time
- benchmarking audit and cost information
-

•

## Quantitative Principles of Computer Design and Analysis

1. **Parallelism**
2. **Locality**
3. **Common Case**
4. **Amdahl's Law** expresses the law of diminishing returns. i.e. keep on adding improvements to the same fraction of the computation will yield diminishing incremental improvements in speedup; if an enhancement is only used for a fraction of a task, then you can't speedup the task by more than the reciprical of 1 minus that fraction.

$$\text{Speedup} = \frac{\text{Performance}_{\text{enhancement}}}{\text{Performance}_{\text{no enhancement}}}$$

$$\text{Speedup} = \frac{\text{Exec Time}_{\text{no enhancement}}}{\text{Exec Time}_{\text{enhancement}}}$$

$$\text{Exec Time}_{\text{new}} = \text{Exec Time}_{\text{old}} \times \left( (1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} \right)$$

$$\text{Speedup}_{\text{overall}} = \frac{\text{Exec Time}_{\text{old}}}{\text{Exec Time}_{\text{new}}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

5. **Processor Performance Equation**:

$$\text{CPU time} = \text{CPU clock cycles} \times \text{Clock cycle time}$$

$$\text{CPU time} = \frac{\text{CPU clock cycles}}{\text{Clock rate}}$$

$$\text{CPI} = \frac{\text{CPU clock cycles}}{\text{instruction count (IC)}}$$

$$\text{CPU time} = \text{IC} \times \text{CPI} \times \text{Clock cycle time}$$

$$\text{CPU clock cycles} = \sum_{i=1}^{n} \text{IC}_i \times \text{CPI}_i$$

$$\text{CPU time} = \Big( \sum_{i=1}^{n} \text{IC}_i \times \text{CPI}_i \Big) \times \text{Clock cycle time}$$

$$\text{CPI}_{\text{overall}} = \frac{\sum_{i=1}^{n} \text{IC}_i \times \text{CPI}_i}{\text{IC}} = \sum_{i=1}^{n} \frac{\text{IC}_i}{\text{IC}} \times \text{CPI}_i$$

- Clock cycle time depends on hardware technology and organization
- CPI depends on organization and instruction set architecture
- IC depends on instruction set architecture and compiler technology.

The three parameters are interdependent, so you can't simply change one and hope to simply improve CPU time, but many performance improvement techniques to one component can have small and predictable impacts on the other two.

- Energy Efficient techniques (dynamic voltage frequency scaling, overclocking, etc) can vary clock speed as programs are being measured, making performance equations hard to use.
- Performance and energy generally correlate, so lowering exec time can also save energy.

# 2 Memory Hierarchy

## Readings

- Chapter 2, 5, Appendix B

Caches

3 Microarchitecture and the Pipeline

4 Security from the Hardware/Systems Perspective

5 Multicore and Heterogeneous Systems

6 Power and Energy

7 Section

8 Appendix

# References

[1] John L. Hennessy and David A. Patterson. 2017. Computer Architecture, Sixth Edition: A Quantitative Approach (6th. ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.