High Performance Programming and Systems

Troels Henriksen, Philippe Bonnet

Agenda

What are computer systems?

Motivation

- Why do computer numbers behave strangely?
- Why are some languages faster than others?
- How do we access memory efficiently?
- What does “efficiency” or “performance” even mean?
- Course perspective

Course organisation

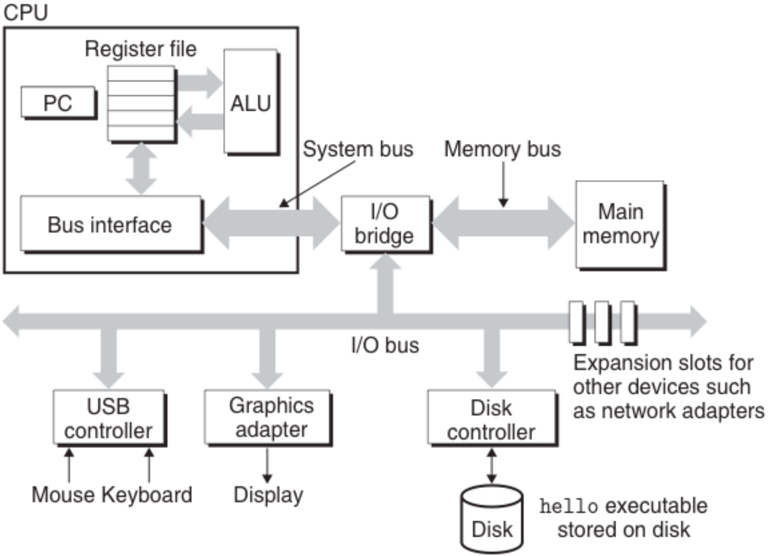
What are computer systems?

Motivation

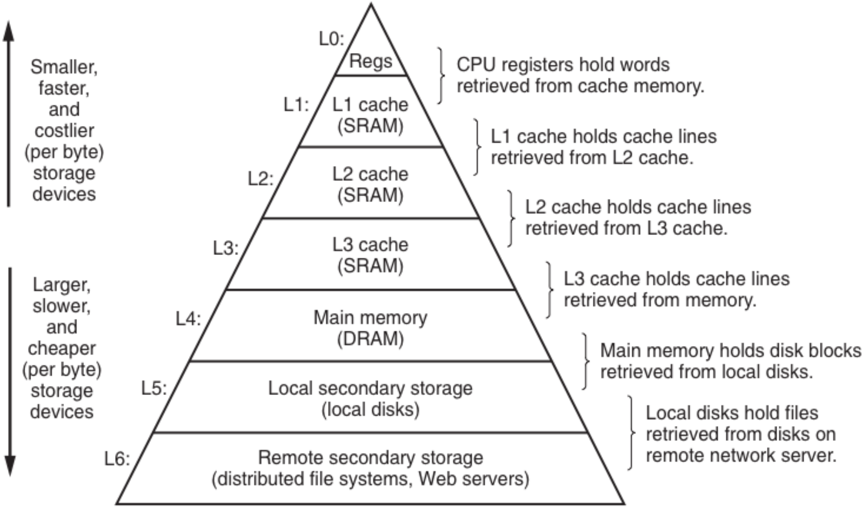
- Why do computer numbers behave strangely?
- Why are some languages faster than others?
- How do we access memory efficiently?
- What does “efficiency” or “performance” even mean?
- Course perspective

Course organisation

Computer System: Hardware



Computer System: Memory Hierarchy



Smaller, faster, and costlier (per byte) storage devices

Larger, slower, and cheaper (per byte) storage devices

L0: Regs

L1: L1 cache (SRAM)

L2: L2 cache (SRAM)

L3: L3 cache (SRAM)

L4: Main memory (DRAM)

L5: Local secondary storage (local disks)

L6: Remote secondary storage (distributed file systems, Web servers)

CPU registers hold words retrieved from cache memory.

L1 cache holds cache lines retrieved from L2 cache.

L2 cache holds cache lines retrieved from L3 cache.

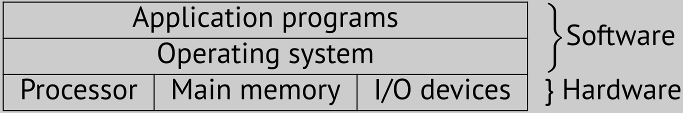
L3 cache holds cache lines retrieved from memory.

Main memory holds disk blocks retrieved from local disks.

Local disks hold files retrieved from disks on remote network server.

Computer System: Abstraction Layers

Layered view of computer system



Application programs

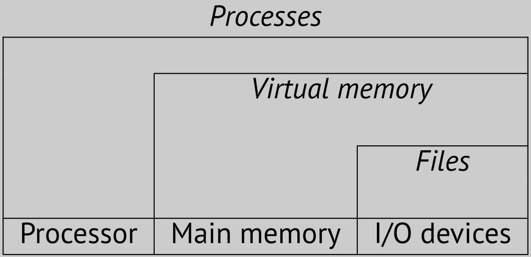
Operating system

Processor | Main memory | I/O devices

Software

Hardware

Abstractions provided by operating system



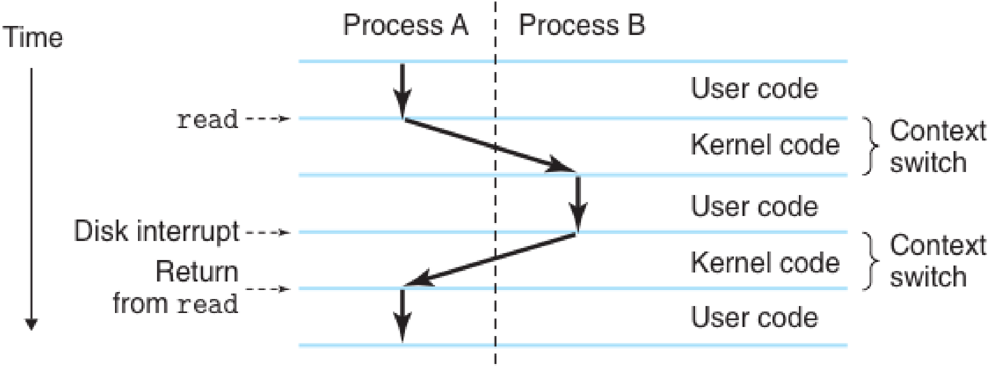
Processes

Virtual memory

Files

Processor | Main memory | I/O devices

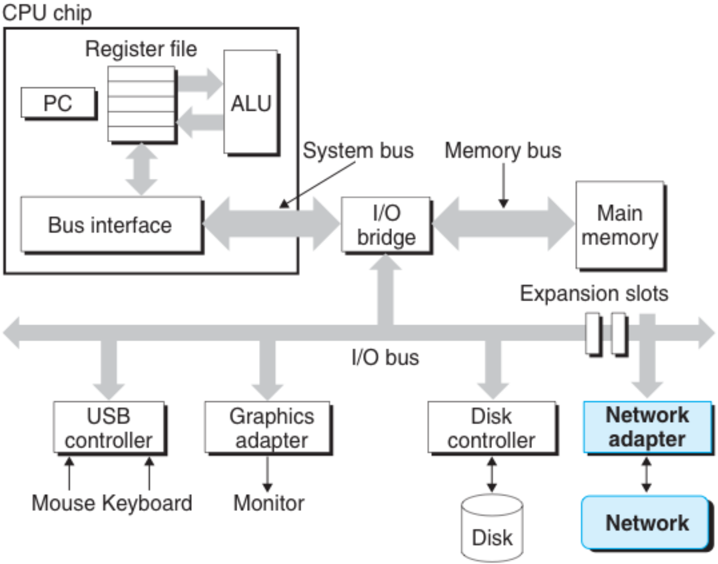
Processes



- Time-sharing via *context switching*.
- Each process has the illusion of exclusive access to the system.

The network

- Networks are systems of systems.
- How do they communicate?
- How are they made robust?



What are computer systems?

Motivation

Why do computer numbers behave strangely?
Why are some languages faster than others?
How do we access memory efficiently?
What does “efficiency” or “performance” even mean?
Course perspective

Course organisation

Why do we force you to study this?

- **This course exists for two reasons**
  1. The bureaucratic reason: For DS/DatØk-students to be eligible for the MSc in CS, you must have been taught certain topics within computer systems.

Why do we force you to study this?

- **This course exists for two reasons**
  1. The bureaucratic reason: For DS/DatØk-students to be eligible for the MSc in CS, you must have been taught certain topics within computer systems.
  2. A better reason: Because data analysis and simulation is often performance-critical, and performance depends on understanding the abstraction layers you use.

Why do we force you to study this?

- **This course exists for two reasons**
  1. The bureaucratic reason: For DS/DatØk-students to be eligible for the MSc in CS, you must have been taught certain topics within computer systems.
  2. A better reason: Because data analysis and simulation is often performance-critical, and performance depends on understanding the abstraction layers you use.

Questions we can answer at the end of the course

- Why do computer numbers behave strangely?
- Why are some languages faster than others?
- How do we access data efficiently?
- How do we take advantage of parallel computers?
- How can programs on different computers communicate?
- What does “efficiency” or “performance” even mean, and how do we quantify it?
- **What is the difference between representation and interpretation?**

What are computer systems?

Motivation

- Why do computer numbers behave strangely?
- Why are some languages faster than others?
- How do we access memory efficiently?
- What does “efficiency” or “performance” even mean?
- Course perspective

Course organisation

`int`s are not integers, `float`s are not reals

- Is $x^2 \geq 0$?
  - `float`: Yes!
  - `int`:
    - $40000 \times 40000 \rightarrow 1600000000$
    - $50000 \times 50000 \rightarrow -1794967296$
- Is $(x + y) + z = x + (y + z)$?
  - `int`: Yes!
  - `float`:
    - $(10^{20} + -10^{20}) + 3.14 \rightarrow 3.14$
    - $10^{20} + (-10^{20} + 3.14) \rightarrow 0.00$

Computer arithmetic

- Does not generate random values:
  - **Deterministic rules.**
  - Useful mathematical properties.
  - …but not always intuitive.
- Finiteness of representation loses some usual mathematical properties:
  - `int`s are rings: Commutativity, associativity, distributivity.
  - `float`s are ordered: Monotonicity, signs.
    - Well, almost…
- What do we gain?
  - + **Performance:** hardware is *fast* at working with fixed-size data.
  - + If we understand the rules, we can write very efficient (and correct!) code.
  - + Can build slower but “more mathematical” numbers on top, as an abstraction layer.

What are computer systems?

Motivation

- Why do computer numbers behave strangely?
- Why are some languages faster than others?
- How do we access memory efficiently?
- What does “efficiency” or “performance” even mean?
- Course perspective

Course organisation

Summing in Python versus C

```c
double s = 0;
int i = 0;
while (i < n) {
  s += i;
  i += 1;
}
```

9ms for $n = 10^7$.

```python
s = 0
i = 0
while i < n:
    s += i
    i += 1
```

1604ms for $n = 10^7$.

- Why this enormous difference?
  - Computers execute *machine code instructions*.
  - C is *compiled to machine code*, Python is *interpreted by another program*.
- Is a C program always vastly faster than a Python program?
  - No: libraries like Numpy let Python perform well.
    - ...so how do they work?
  - Choice of language is not the only thing that matters.

What are computer systems?

Motivation

- Why do computer numbers behave strangely?
- Why are some languages faster than others?
- How do we access memory efficiently?
- What does “efficiency” or “performance” even mean?
- Course perspective

Course organisation

Memory system performance example—2.0 GHz Intel Core i7 Haswell

```
void copyij(int src[2048][2048],
            int dst[2048][2048]) {
  int i,j;
  for (i = 0; i < 2048; i++)
    for (j = 0; j < 2048; j++)
      dst[i][j] = src[i][j];
}
```

4.3ms

```
void copyij(int src[2048][2048],
            int dst[2048][2048]) {
  int i,j;
  for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
      dst[i][j] = src[i][j];
}
```

81.8ms

- Performance depends on access pattern.
- C lays out arrays in *row-major order*:

```
src[0][0], src[0][1], ..., src[0][2047], src[1][0], src[1][1], ...
```

  - **Left** traverses elements with stride 1.
  - **Right** traverses elements with stride 2048.
- **Locality is key to performance!**

What are computer systems?

Motivation

- Why do computer numbers behave strangely?
- Why are some languages faster than others?
- How do we access memory efficiently?
- What does “efficiency” or “performance” even mean?
- Course perspective

Course organisation

Quantifying performance

- **If you want to improve something, you need to be able to measure it.**
  - (At least when it comes to machines; don’t treat people like machines.)
- Previously:
  - C: 9ms
  - Python: 1604ms
  - Clearly “the C program is faster”, but how do we report this in a standard way?
- **Different kinds of performance:**
  - **Latency** – how fast you respond or complete a task.
  - **Throughput** – how many tasks you complete per time unit.
  - **Discussion:** compare latency and throughput of cargo truck and cargo ship.
- **Scalability**
  - How does our system or program behave when we change the workload or run it on a faster machine?

What are computer systems?

Motivation

- Why do computer numbers behave strangely?
- Why are some languages faster than others?
- How do we access memory efficiently?
- What does “efficiency” or “performance” even mean?
- Course perspective

Course organisation

Course perspective

- **HPPS is programmer-centric.**
  - By knowing more about the system, one becomes a more effective programmer.
    - + Faster programs.
    - + More reliable programs.
  - Many of these properties are **universal**.
    - We teach you many low-level details...
    - ...but the concepts (e.g. locality) exist at every level.

What are computer systems?

Motivation

- Why do computer numbers behave strangely?
- Why are some languages faster than others?
- How do we access memory efficiently?
- What does “efficiency” or “performance” even mean?
- Course perspective

Course organisation

Course structure

Textbooks

HPPS: HPPS course notes (mandatory, free)

JG: Modern C (optional, free)

… feel free to replace JG with some other C textbook.

Assignments

- **Five** weekly group assignments.
- Preferably **three students per group**.
- **Graded** with 0-4 points.
- **No resubmissions.**
- Exam qualification: **at least 10 points** and **at least one point** per assignment.
- **First assignment available now.**

Physical teaching

Lecture:	Tuesday 10:00-12:00 (Aud 04, HCØ)
Exercises:	Thursday 10:00-12:00 (locations below)
Lecture:	Thursday 13:00-15:00 (Aud 02, HCØ)
Exercises:	Thursday 15:00-17:00 (locations below)

Exercise locations

Morning:

- Karnapsalen, Nørre Alle 53
- 4-0-02, Ole Maaløes Vej 5, Biocenter
- NBB 2.0.G.064/070, Jagtvej 155

Afternoon:

- 4-0-02, Ole Maaløes Vej 5, Biocenter
- 4-0-13, Ole Maaløes Vej 5, Biocenter

We may adjust this during the course.

Resources

Absalon

- Used for handins, announcements, and discussion forum.

Material

- `https://github.com/diku-dk/hpps-e2025-pub/`
- Handout of all material (info, assignments, exercises, slides).
- You do *not* need to use Git; just treat it as a website.

Discord

- Invite link on website.
- A persistent online exercise class.

Videos

- We made some in previous years that we will link when relevant.

Teachers

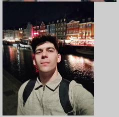
Troels Henriksen: Operating systems, parallelism, networks

Philippe Bonnet: Machine architecture, data representation

Teaching assistants (TAs)

Kevin Mark Lock

August Rehm



Márk Viski

Sofie Larsen

Carl August Gjerris Hartmann

Groups

Size

2-3 student advised. 1 can be accepted but not recommended.

- Sign up for classes with your group-mates on Absalon.
- If you need one or more members:
  - Post on Discord
  - Post on the Absalon discussion board
  - Ask a TA
  - **Do it as soon as possible!**

Assignment rules

Core rule

Each group must construct their own solution.

This means

- You can talk with other people about the assignments: Teachers, TAs, other students, etc.

- You cannot share written code with other groups.
- You are not allowed to use code that you did not write yourself, without proper citation.
- You cannot share written text with other groups.
- You are not allowed to use text of material, without proper citation
  - This also includes material provided by the course.

Assignments versus exercises

- Note! Both are important.
- The exercises
  - Most exercises essentially have you develop the code handed out for the assignment.
  - For the assignment, can use either your own code or the assignment code handout.
- **Assignments assume that you have solved the related exercises.**
  - Assignment code handout may be hard to understand otherwise.

Tools

- C compiler – `gcc` or `clang`.
- C debugger – `gdb` on Linux or `lldb` on macOS.
- You can also install all tools on your laptop
  - Linux: available through your package manager.
  - macOS: available through Homebrew.
  - Windows: Windows Subsystem for Linux.

- Set up your tool chain
  - We recommend using Git to share code and reports in your group.
  - Sign-up at GitHub today and apply for the *Student Developer Pack*.
  - `https://education.github.com/`.

Exam

- **Three-day individual take home exam**.
  - Designed for 20 hours of labour; the exam office keeps changing how many days this is actually spread over.
    - This year it seems to be “51 hours”, so presumably a three-day exam.
- Intended to be very similar to assignments:
  - Analyse a program based on what you have learnt.
  - Rewrite it to make it faster.
  - Write a (short!) report.
- The course curriculum is the exercises, assignments and reading material.
- Examples of previous exams are available on the course website.

QUESTIONS?