High Performance Programming and Systems

Troels Henriksen, David Marchant

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?

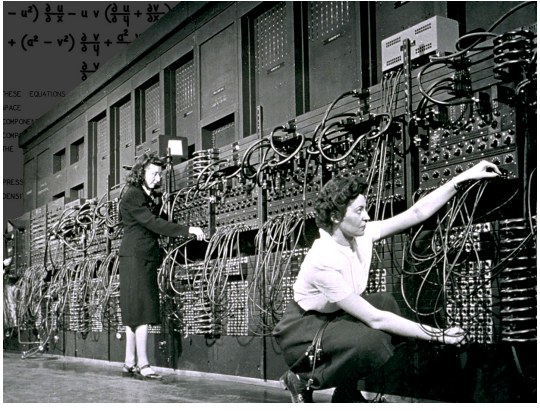
Motivation

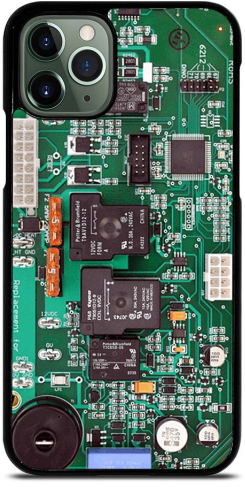
- Why do computer numbers behave strangely?
- Why are some languages faster than others?
- How do we access memory efficiently?
- What does “efficiency” or “performance” even mean?
- Course perspective

Course organisation

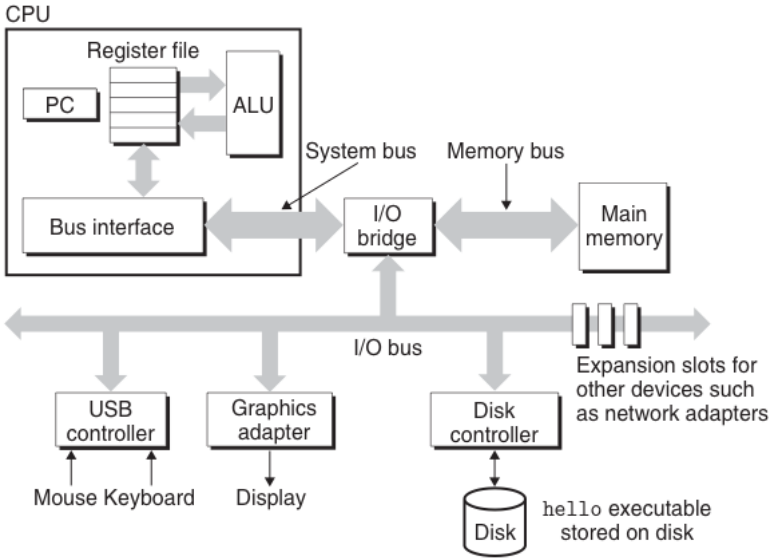
What is a computer system for you?

- **5 minutes!** What is a “computer” to you? What does it do?

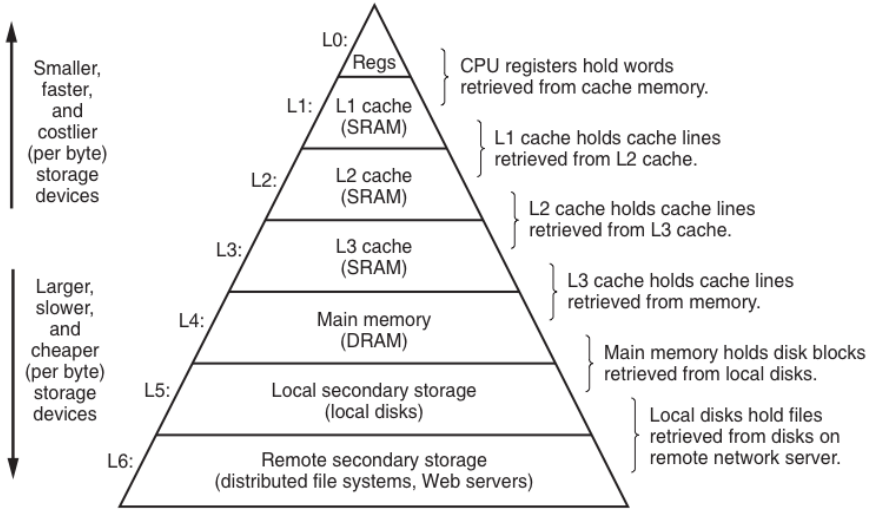




Computer System: Hardware

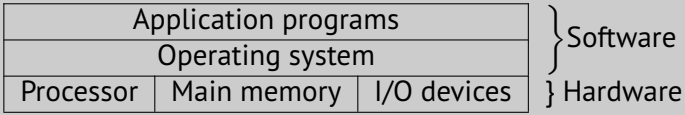


Computer System: Memory Hierarchy

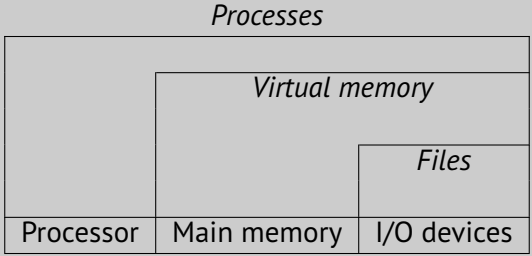


Computer System: Abstraction Layers

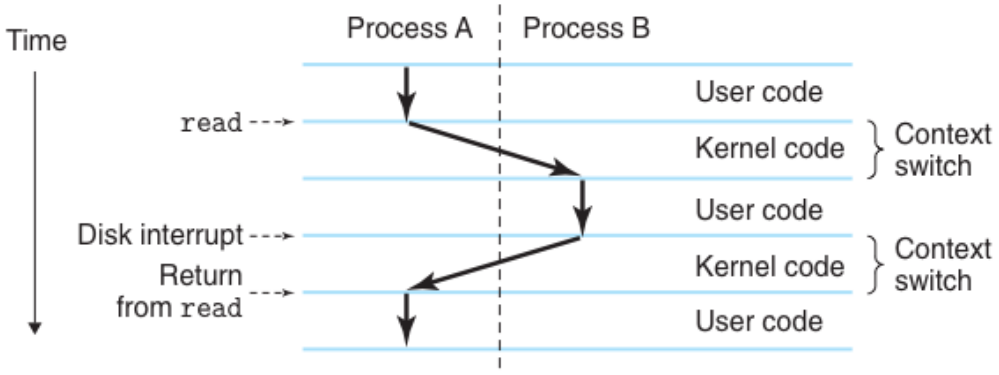
Layered view of computer system



Abstractions provided by operating system



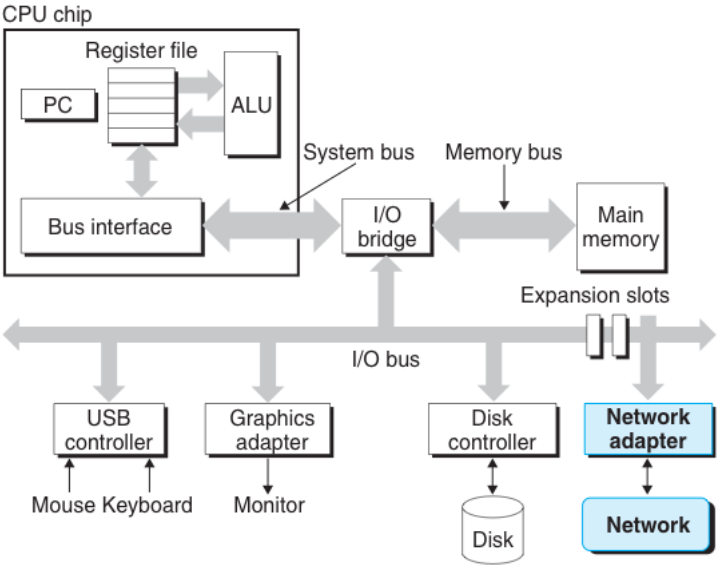
Processes



- 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 Master's Programme in CS, you must have been taught computer systems and network programming.
  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?
- Why are network programs often fragile?
- How do we access data efficiently?
- What does "efficiency" or "performance" even mean?

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)

Assignments

- **Five** 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.

Physical teaching

Lecture:	Tuesday 10:00-12:00 (Aud 05, HCØ)
Exercises:	Thursday 10:00-12:00 (locations below)
Lecture:	Thursday 13:00-15:00 (Aud 05, HCØ)
Exercises:	Thursday 15:00-17:00 (locations below)

Exercise locations

Hold 1+2:	1-0-04 (DIKU)
Hold 3+4:	1-0-37 (DIKU)
Hold 5+6:	3-0-25 (DIKU)

Resources

Absalon

- Used for handins, announcements, and discussion forum.

Material

- `https://github.com/diku-dk/hpps-e2022-pub/`
- Handout of all material (info, assignments, exercises, slides).
- You do *not* need to use Git; just treat it as a website.

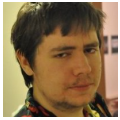
Discord

- Invite link: `https://discord.gg/rEs865W5SA`
- Real-time discussion.

Videos

- We made some last year that we will link when relevant.
- BOH authors have recorded their lectures, which might be interesting: `https://www.youtube.com/playlist?list=PLmBgoRqEQCWy58EIwLSWwMPfkwLOLRM5R`

Teachers



Troels Henriksen: Machine architecture, Operating systems

David Gray Marchant: Network programming, concurrency

Teaching assistants (TAs)

- Axel Højmark
- Gunnar Magnussen
- Mikkel Willén
- Jacob Christian Herbst
- Jonatan Ruiz-Molsgaard
- Rasmus Damgaard Nielsen

TAs may also help with:

- Group members.

Groups

Size

2-3 student advised. 1 can be accepted but not recommended. More than 3 is only allowed under special circumstances.

- Sign up for classes with your group-mates on Absalon.
- If you need one or more members:
  - Look for announcements with details.
  - TAs will facilitate.

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 train the theory you will need in the assignments.
  - Some exercises essentially have you develop the code handed out for the assignment.
- Assignments assume that you have solved the related exercises.

Tools

- C compiler – `gcc` or `clang`.
- C debugger – `gdb` on Linux or `lldb` on macOS.
  - Special setup of VirtualBox if needed. Can get help at exercises
- You can also install most tools on you laptop
  - Linux: available through your package manager.
  - macOS: available through Homebrew.
  - Windows: Windows Subsystem for Linux.

- Set up your tool chain
  - recommended 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/`

- Unix software is available on GitHub.

Post-lecture self-reflection

- Due to schema group placement, we have no exercise classes after tuesday lectures.
- Instead we have *self-reflection exercises*
  - Small questions you can solve on your own to test your understanding of the material.
  - Or set up the tools that might be needed on Thursday.
  - No TAs, but you can try asking for help on Absalon and Discord—and feel free to help each other!
- The self-reflection exercises for today are on basic C programming (the actual point is ensuring you can compile C programs).

Exam

- **Five-day individual take home exam**.
- (Not full-time work for five days.)
- 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.
- New exam form, so **no previous exams available**.

QUESTIONS?