# Data-oriented Design when Performance matters

Abdullah Barhoum

Beiträge zu Software Engineering
WS19/20

# Demo

# Before we start

- Feel free to interrupt
- Food for thought and discussion

# Contents

- Performance
- Caches
- OoD and Performance
- DoD
- Conclusions

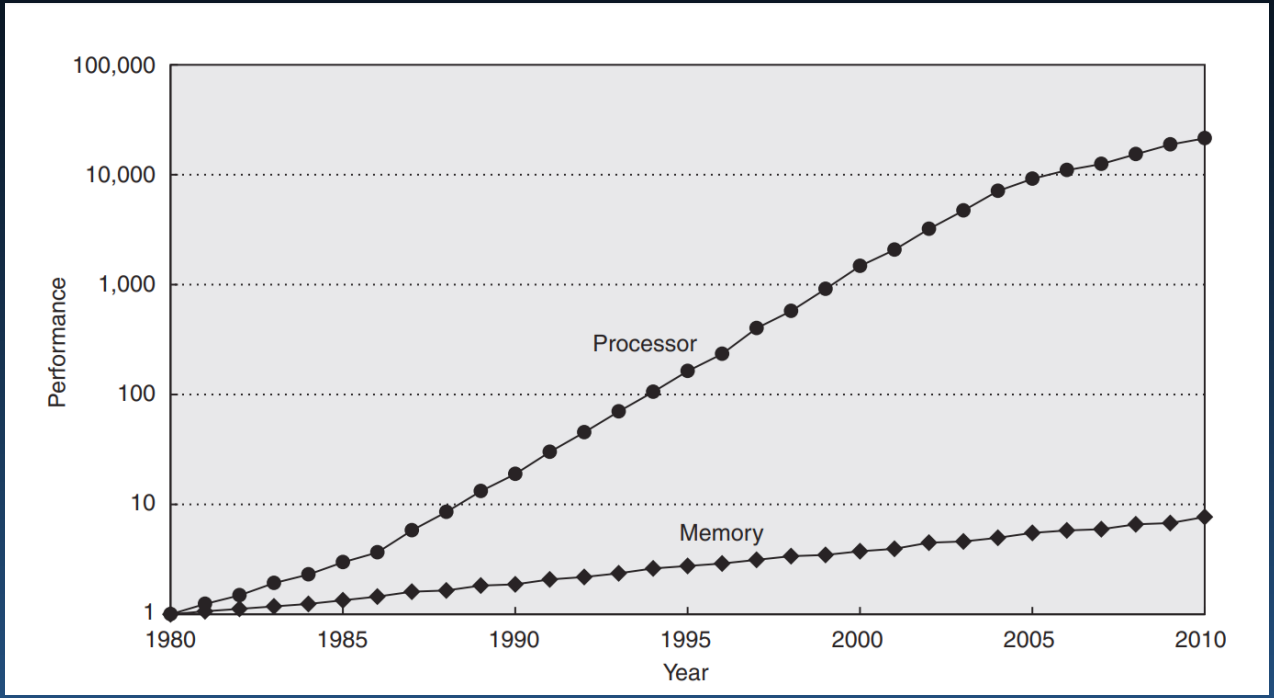
# Performance?

- Your take?

# Performance?

- Efficiency with Algorithms
- Performance with Data Structures

# Performance Gap



# Bottleneck

- Data access from memory
- Not thinking about the Cache

# Caches

- Types:
  - Data Cache (D-Cache)
  - Instruction Cache (I-Cache)
  - Translation Lookaside Buffer (TLB)
- Hierarchies:
  - L1
  - L2
  - L3?

# Intel Core i79xx

- 4 cores, 8 hardware threads
- Hierarchies:
  - L1 (32KB I-Cache, 32KB D-Cache), 4 cycle latency
  - L2 (256KB I/D-Cache), 11 cycle latency
  - L3 (8MB I/D-Cache for all cores), 39 cycle latency
    - Everything in L1 and L2 has to be in L3
  - RAM (Varies), at least 107 cycle latency


L1
L2
RAM

# Cache Lines

- The ‘currency’ between RAM and CPU
- 64 bytes in the aforementioned Processor
- Prefetching!

# Conclusions

- Small ≡ fast
- Locality counts, stay in the cache as much as possible both for instructions and for data.
- Predictable access patterns help with prefetching and minimize stalling.

# How OOD hurts Performance

- D-Cache Misses
- I-Cache Misses

# Data-oriented Design (DoD)

- Mike Acton

- If you don’t understand the data, you don’t understand the problem.

- Different problems require different solutions.

  If you have different data, you have a different problem

- If you don’t understand the cost of solving the problem, you don’t understand the problem.

  And if you don’t understand the hardware, you can’t reason about the cost of solving the problem.

- Solving problems you probably don’t have, creates more problems you definitely do.
- You can’t “Future Proof”.

- Where there is one, there are many. Try looking on the time axis.
- The more context you have, the better you can make the solution. Don’t throw away data you need.
- You want to know more about where, when and how every piece of data is going to be used.

- Software does not run in a magic fairy aether powered by the fevered dreams of CS PhDs.

  Software is real and runs on real hardware to solve a real problem.

# “Programming Misconceptions”

- Wrong:
  - Code is more important than data
  - Code should be designed around model of the world
- Correct:
  - Data is just as important if not more
  - Code should solve the problem, not over-complicate it

# Examples

Understanding the cost of solving the problem

# From OoD to DoD

- The OoD way
- Objects as a composition of Components
- Objects as a container of Components' references
- Component-Systems

# DoD Advantages

- Cache utilization & performance
- Parallelization

# DoD Disadvantages

- DoD vs
  - Programmer
  - Programming Language
  - OoD
- Lack of flexibility

# Takeaways

- Optimal cache usage is important for performance
  - DoD can help with that
- OoD vs DoD, which is better?
  - No one fucking cares
  - Whatever solves YOUR PROBLEM

*Das Gehirn ist keine Seife, es wird nicht weniger wenn man es verwendet.*

- Lisa Fitz

# References


- Scott Meyers, CPU Caches and Why You Care
  https://www.youtube.com/watch?v=WDIkqP4JbkE

- Mike Acton, Data-Oriented Design and C++
  https://www.youtube.com/watch?v=rX0ItVEVjHc

- Chandler Carruth, Efficiency through Algorithms, Performance through Data Structures
  https://www.youtube.com/watch?v=fHNmRkzxHWs

- John L Hennessy and David A Patterson. Computer architecture: a quantitative approach. Elsevier, 2011

# Might be interesting

- Things that Matter - Scott Meyers
  https://www.youtube.com/watch?v=RT46MpK39rQ

- Marvel's Spider-Man: A Technical Postmortem
  https://youtu.be/KDhKyIZd3O8

- A Programming language for games
  https://www.youtube.com/playlist?list=PLmV5I2fxaiCKfxMBrNsU1kgKJXD3PkyxO