Project
Algorithm Engineering

Jens K. Mueller

jkm@informatik.uni-jena.de

Department of Mathematics and Computer Science
Friedrich-Schiller-University Jena

Wednesday 9th April, 2014

Organization & Introduction

Administrative

- 6 LP course (1 LP is about 30 h)
  180 h/14 week $\approx$ 13 h/week (on average)
- Mostly self-study (only 2 h/week in class)
- **Wednesday 2.15pm** in **EAP2 3325** or
  Thursday 2.15pm in **EAP2 3325**

Course web page at
http://theinf2.informatik.uni-jena.de/Lectures/Algorithm Engineering (project).html

Lecture Part

- Toy problems for illustration
- Follow best practices
- Show tools
- Rubber ducking
- Provide advice

Your Part

- Choose/propose two projects
- Push the project on your own
- Report on your progress
- Reflect and understand what you did
- Do measurements and plot results

Rules

- Use available resources but give credit where credit is due
- Make sure you understand/know what you are doing
- Cheating will be reported and results in failing this course

Examination

- Oral exam (very near after the course's end)
- About your projects
- To evaluate your progress and solution

Algorithm Engineering

Gap between theoretical analysis and empirical performance

- Constants in big O notation, memory hierarchy, NUMA architectures, advanced CPU instructions
- Asymptotic (running) time complexity
- Typically worst case analysis
- Problem properties not exploited
- NP-hard problems may be solvable for practical applications

Reduce the gap by addressing the (simplifying/impractical) assumptions (simplified machine model, big O notation, ...).

Summary

- Semantically equivalent programs may not have equal performance
- Constants matter in practical applications
- Squeeze out the hardware

Beware!

We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil.

Summary

- Semantically equivalent programs may not have equal performance
- Constants matter in practical applications
- Squeeze out the hardware

Beware!

We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%. A good programmer will not be lulled into complacency by such reasoning, he will be wise to look carefully at the critical code; but only after that code has been identified.

Donald Knuth

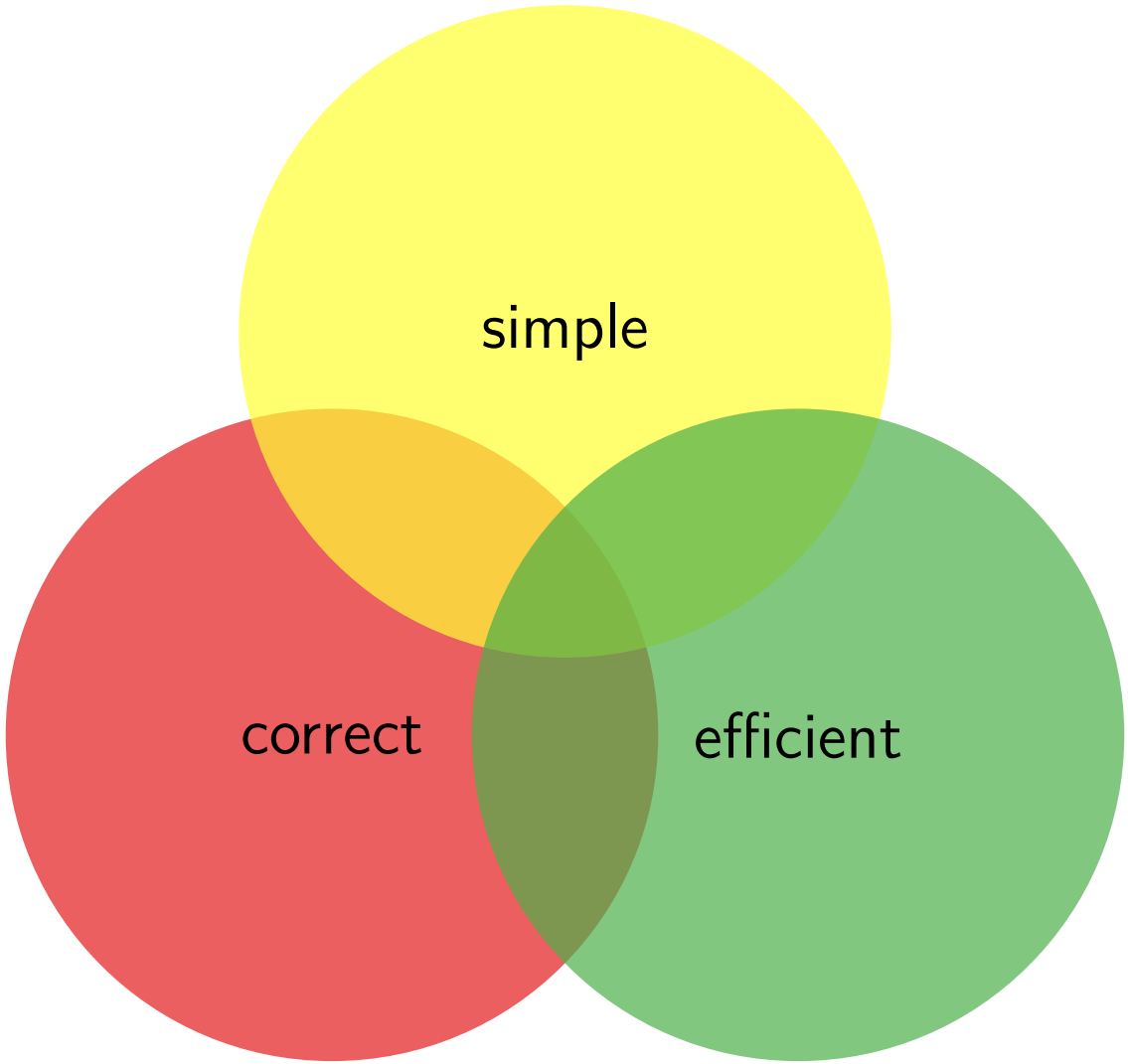
What I want you to take away

1. A strategy to implement/engineer algorithms
2. What and how to exploit hardware for algorithm engineering

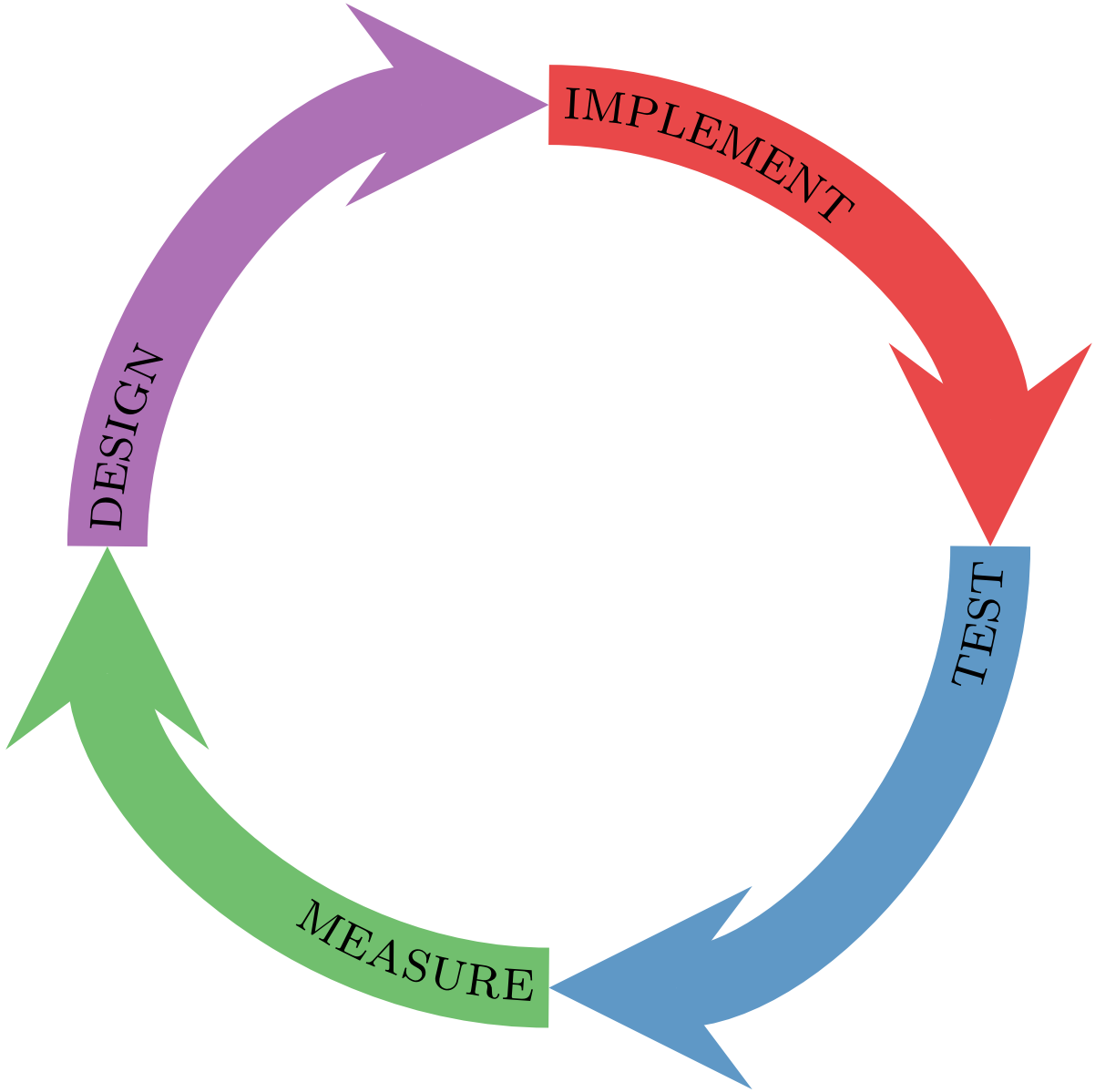
A bit technical

- Nitty-gritty details
- Look at assembly code
- CPU architecture

Paths of Glory



Each step follows a Cycle



Tools and Techniques

- Testing
- Profiling & Measuring
- Debugging

Your Environment

- Pick a language (C, C++, or D with ≥ 2 compilers)
- Choose whatever editor/IDE you're comfortable with
- Setup to build, compile and reproduce your results
- Record your changes (progress and dead ends)

Example D tools

- Install D latest compilers (dmd, gdc, ldc)
  http://wiki.dlang.org/Compilers
- Editors
  http://wiki.dlang.org/Editors
- IDEs
  http://wiki.dlang.org/IDEs
  - DDT (Eclipse-based) installation
  - Visual D (Visual Studio) installation
  - Mono-D installation
  - Code::Blocks

Projects

1.
   - BLAS level 1 (vector operations)
   - BLAS level 2 (vector-matrix operations)
   - ...
2.
   - BLAS level 3 (matrix-matrix operations)
   - Solving systems of linear equations
   - Eigenvalue problem (also SVD)
   - Matrix decomposition (LU, QR, Cholesky and Schur)
   - Sparse BLAS
   - Transforms
   - ...