Project
Algorithm Engineering

Jens K. Mueller

jkm@informatik.uni-jena.de

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

Wednesday 9th July, 2014

Summary

Guide

1. Start simple
2. Follow the path between premature optimization/pessimation (readability is king)
3. Ensure correctness through testing
4. Incrementally move forward guided by measurements
5. Optimize and verify assembly
6. Prefer high-level optimization over low-level ones
7. Memory vs. CPU-bound algorithm
   Average #operations per in-/output

$$\frac{\text{number of operations}}{\text{size of in- and output}}$$

Low-Level Optimizations

- Code motion
- Loop unrolling
- Reassociation
- Vectorization
  Efficiency

$$\frac{\text{number of operations in scalar code}}{\text{number of operations in vectorized code}}$$

- Strength reduction

Compiler Struggle

- Memory aliasing
- Proof absence of side-effects
- Needs to be conservative (preserve observed behavior) but may exploit undefined behavior
- Bad at making choices

Consider

- Memory hierarchy
- Critical path
- SoA over AoS (if applicable)

Algorithmic Optimizations

- Good choice of algorithm/data structure most important
- Asymptotic complexity ignores constants and optimizes for infinite input size
- Care about realistic worst cases in your setting
- Non-optimal algorithm may be better on small input

Consider

1. Know your input
2. Data representation (data structures)
3. Algorithms
4. Vectorization friendly algorithm

Program Performance Metrics

- Running time over increasing input sizes
- FLOP/s
- cycle/element
- Cache misses, branch misprediction

Limits

- Instruction latency and throughput
- Theoretical performance and bandwidth

Exam

- Send your code
- Optionally prepare introduction (problem, approaches, measurements, plots)
- Questions related to your project (and the lecture)

References

Randal E. Bryant and David R. O'Hallaron. Computer Systems: A Programmer's Perspective. 2nd. USA: Addison-Wesley, 2010. ISBN: 0136108040, 9780136108047.

Markus Püschel. "How to Write Fast Numerical Code". URL: `http://www.inf.ethz.ch/personal/markusp/teaching/263-2300-ETH-spring13/course.html` (visited on 04/02/2014).