

Algorithm Engineering

Jens K. Mueller

`jkm@informatik.uni-jena.de`

Department of Mathematics and Computer Science
Friedrich Schiller University Jena

Monday 12th January, 2015

Optimizing Compiler

The biggest speedup you'll ever get with a program will be when you first get it working.

JOHN OSTERHOUT (creator of Tcl and Tk)

A Compiler's Job

- ▶ High level language
- ▶ (Internal representation)
- ▶ Assembly
- ▶ Machine code

Disassembler

- ▶ Stop before assembling
- ▶ `$ objdump -D <binary>`

Optimizing Compiler

- ▶ Apply **safe** optimizations to use less running time or less space

Optimizations

- ▶ Code motion
- ▶ Inlining
- ▶ Loop Unrolling
- ▶ Vectorization
- ▶ ...

Optimizations

- ▶ Interact with each other
- ▶ Must be safe (do not alter observable program behavior)
- ▶ May use (language) undefined behavior

Optimization blockers

- ▶ Memory aliasing
- ▶ Inability to infer purity (free of side effects and same arguments)

CPU Architecture

- ▶ Superscalar
- ▶ Out-of-order
- ▶ Speculative (branch prediction)
- ▶ Special instructions
- ▶ Latency and issue time of instructions

Guiding the compiler

Common Optimizations

- ▶ Move code
- ▶ Reducing function calls
- ▶ Reduce memory references

Move Code

- ▶ Check inner loop in assembly
- ▶ Massage code or move manually

Reducing Calls

- ▶ Check assembly for calls
- ▶ Inlining

Memory References

- ▶ Identify redundant memory references
- ▶ Eliminate unneeded checks (bounds)

Tips

- ▶ Be careful when sacrificing code readability
- ▶ Enable optimizations
- ▶ Test different compilers
- ▶ Read assembly code

Cycles per Element (CPE)

- ▶ rdtsc
- ▶ Performance counters
- ▶ Least squares (to accommodate measuring code overhead)