Small Guide to Writing Fast Code

Markus Püschel
Carnegie Mellon University
www.ece.cmu.edu/~pueschel

Guideline for writing fast Code (I)

- Avoid obvious mistakes
  - Know the available algorithms: use good algorithms
  - Precompute once were possible (e.g., constants)
  - Give the compiler a chance: write simple code, avoid complicated data structures
  - Understand were the runtime is wasted
    - code profiling
    - optimize only code parts that matter
  - Compiler
    - Use vendor compiler if possible
    - Try different flags, -O3 is not always best
- Optimization for caches
  - Recursive is better then iterative
  - Understand your code in terms of cache behavior and try to improve
  - Know your cache size and maybe other parameter

Guideline for writing fast Code (II)

- Basic block optimization
  - For the innermost kernels use unrolled code: no loops, recursive calls or other control structures
  - Order instructions for register locality and/or instruction parallelism; scalar replacement for variables being reused; other optimizations
  - Maybe: check assembly code
- Adaptivity through search over alternatives
  - Accept that you can't know the right answers for all choices
  - Search over a relevant subset of possible algorithms and/or implementation options
- After optimization check whether you still use the right algorithm

How to Write Good SIMD Vector Code

- Take the “right” algorithm and the “right” data structures
  - Fine grain parallelism
  - Correct alignment in memory
  - Contiguous arrays
- Use a good compiler (e. g., vendor compiler)
- First: Try compiler vectorization
  - Right options, pragmas and dynamic memory functions
    (Inform compiler about data alignment, loop independence,…)
  - Check generated assembly code *and* runtime
- If necessary: Write vector code yourself
  - Most expensive subroutine first
  - Use intrinsics, no (inline) assembly
  - Important: Understand the ISA

Further Reading

- Check out the Proceedings of the IEEE special issue on "Program Generation, Optimization, and Platform Adaptation," Feb. 2005, 93(2) and the references therein.