CS 516 Compilers and Programming Languages II

Compiler Optimizations
Implications

• Use an intermediate representation (IR)
• Front end maps legal source code into IR
• Back end maps IR into target machine code
• Extension: multiple front ends & multiple passes (better code)

Typically, front end is $O(n)$ or $O(n \log n)$, while back end is NPC
Traditional Three-pass Compiler

Code Improvement (or Optimization)

- Analyzes IR and rewrites (or transforms) IR
- Primary goal is to reduce running time of the compiled code
  → May also improve space, power dissipation, energy consumption, ...
- Must preserve “meaning” of the code (may include approximations, i.e., quality of outcomes tradeoffs)
  → Measured by values of named variables or produced output
The Optimizer (or Middle End)

Typical Transformations
- Discover & propagate some constant value
- Move a computation to a less frequently executed place
- Specialize some computation based on context
- Discover a redundant computation & remove it
- Remove useless or unreachable code
- Encode an idiom in some particularly efficient form

Modern optimizers are structured as a series of passes
Constant propagation/folding

Dead code elimination

CSE: common subexpression elimination

Invariant code motion

Strength reduction, idioms recognition

Procedure inlining
Read paper posted on our web site

More detailed discussion of compile-time performance and power/energy optimizations