Abstraction Cost and Optimization

Johan Engelen

LDC team

https://johanengelen.github.io

Outline

- Optimization and “abstraction cost”
  - Cost of a function call
- Measuring performance
  - common pitfall: compiler is given too much info
- Live code examples
  - Inspired by Jason Turner's CppCon 2016 talk
  - D → assembly

Feel free to interrupt me any time for questions or comments

Optimization

- Code transformations
- Reason about code
- What does the language specification say?

```
void foo()
{
  int a = 1;
}
```

```
void foo()
{
}
```

Abstraction cost

- Optimization = generally removes abstraction artifacts
  - Inlining = remove function call (removes function “abstraction”)
- Abstraction cost = (performance with abstraction)
  minus (performance without)
- Zero-cost abstraction = identical code after optimization
- Possible to have negative cost?
  - Yes: templated functions
- Cost may depend on details
  - What is the cost of a function call?

Cost of a function call

- The cost depends on the callee
  - Is the callee inlinable?
  - How many parameters does the callee take?
- Inlined?
  - Yes: zero cost
  - No: cost of call itself
    plus cost of parameter passing

Cost of a function call (2)

- Performance depends on the size of code (amount of instructions), because of memory load and caching
- Inlining of rarely executed calls ~~is~~ ***may*** be bad for performance
  - Note: the inlined code may be smaller than the call itself
- Future optimization? “outlining” of rarely executed code

```
if (almost_never_true) {
    f1(); // inlining = perhaps bad
} else {
    f2(); // inlining = perhaps good
}
```

Compilers

- DMD, GDC, LDC, SDC, …
  - Compile-time performance: DMD
  - Run-time performance: GDC, LDC, SDC
  - This talk: **LDC**
- LDC does not inline functions from another module
  - It doesn't ?!
  - `-enable-cross-module-inlining`
  - Templates
  - Link-time optimization (LTO)
- Be aware that performance of different Phobos/druntime versions may vary a lot

Measurement

- To know the performance of a piece of code, there is only one way: measurement
  - Obtaining good measurements is far from trivial!
- To obtain a deeper understanding: study compiler output
  - LLVM IR (`-output-ll`): easier to understand why optimization does/doesn't happen, but can't see result of register allocation and instruction selection
  - assembly (`-output-s`): actual instructions executed by the CPU
- In this talk: yes, we are going to discuss performance without measuring :-)

Common pitfall

- Compiler is given too much information!
  - the input data
  - the number of loop iterations
  - the exact type of a polymorphic object
  - the body of a function
  - the alignment of data
  - …

https://d.godbolt.org

- Matt Godbolt's Compiler Explorer
  - Matt's blog: https://xania.org/
- Online compilation of D code to assembly
- Write code on the left, see the assembly output on the right
- Easy to try different compilers and compile flags
- Go visit the page and tinker with the code during this talk!

Final remarks...

- If you want to improve the performance of your code
  - Start by measuring, avoid the pitfalls
  - Analyze compiler output to find out what can be improved
    - It pays off to learn LLVM IR, it's much easier to read than assembly

- There is *a lot* of room left for improvements, a few ideas:
  - Improve devirtualization (a membercall clobbers the vptr? come on!)
  - Memory allocations, elide or turn them into stack allocs (LDC already has GC-->stack but needs improvement)
  - Cross module inlining, or just use LTO?
    Ship LDC with LTO Phobos/druntime!
  - `pure` ? `nothrow` ? `immutable` ?