CS738 - Compiler Optimizations

Programming Assignment 1

Shruti Sharma

Oct 2020

1 Task 1

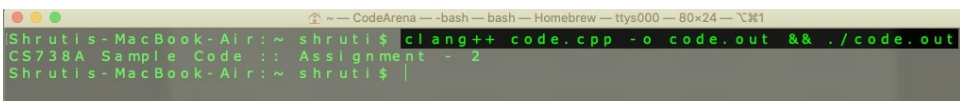

Figure 1: Command

Figure 2: Output

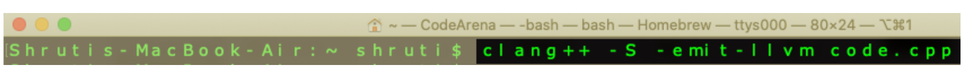

Figure 3: Command with -emit-llvm flag

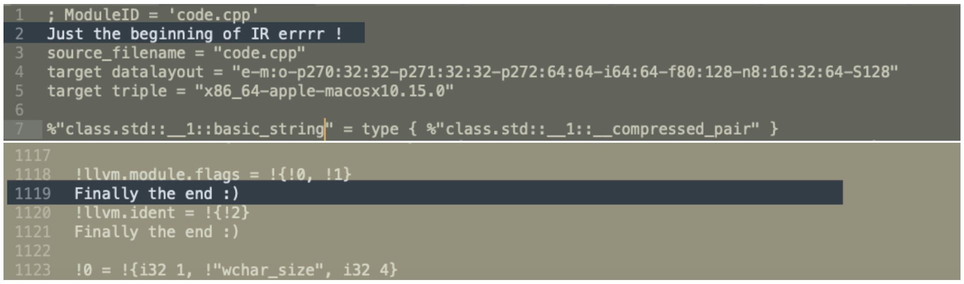

Figure 4: Output

Source file changed - AsmWriter.cpp
AsmWriter.cpp prints LLVM IR as an assembly file. This library implements 'print' family of functions in classes like Module, Function, Value, etc. In-memory representation of those classes is converted to IR strings. This library is used to print LLVM assembly language files to an iostream. The LLVM IR generation part of clang handles conversion of the AST nodes output by the Sema module to the LLVM Intermediate Representation (IR). Historically, this was referred to as "codegen", and the Clang code for this lives in lib/CodeGen. -emit-llvm gives the bitcode which is diassembled using llvm-dis to IR. The task was to inspect this bitcode. An outs() statement can be used as a helper. Just modifying the front-end achieves the goal.

Link to diff: `https://github.com/llvm/llvm-project/commit/a3d5092eda7e57786f96ab018a93f7f9a9271f05`

2 Task 2

2.1 -O0 vs -O1

2.1.1 Example 1

-O1 does **Dead Code Elimination (DCE)** but -O0 performs no such optimization and gives a longer IR compared to its -O1 counterpart.

The source file *(Q1-1.c)* contains dead code, such as a=30 and s=a+b. At optimization level -O0, clang performs no optimization, and therefore generates code for the dead code in the source file. However, at optimization level -O1, clang does not generate code for the dead code in the source file.

lib/Transforms/Scalar/DCE.cpp - This file implements dead instruction elimination and dead code elimination.

2.1.2 Example 2

It is also seen that there are **redundant load and stores** like x=s present in source file *(Q1-2.c)* when the flag -O0 is passed while -O1 removes store instructions which do not affect the result of the program. Flag -O1 eliminates temporary variables and reorders non-redundant stores and loads.

-O1 includes the mem2reg pass. mem2reg particpates in the removal of redundant allocas.

lib/Transforms/Scalar/PromoteMemoryToRegister.cpp - It promotes alloca instructions which only have loads and stores as uses.

2.2 -O1 vs -O2

2.2.1 Example 1

When compiling the source file *(Q2-1.c)* with -O2 optimization, it runs the pass always-inline which **inlines all the function calls** and treats the code as one big function. However, -O1 uses the inline keyword in source file as just a hint but doesn't actually inlines the functions.

2.2.2 Example 2

For source file *(Q2-2.c)*, the globaldce pass, eliminates unreachable internals from the code. It also performs a global value numbering pass that eliminates partially or fully redundant instructions and eliminates redundant load instructions.

lib/CodeGen/GlobalMerge.cpp - Internal globals merging. This pass merges globals with internal linkage into one. This way all the globals which were merged into a biggest one can be addressed using offsets from the same base pointer (no need for separate base pointer for each of the global). Such a transformation can significantly reduce the register pressure when many globals are involved.

2.3 -O2 vs -O3

Using the O3 optimizations may not cause higher performance unless loop and memory access transformations take place. The optimizations may slow down code in some cases compared to O2 optimizations. Here are the proofs for the same.

2.3.1 Example 1 (Q3-1.c)

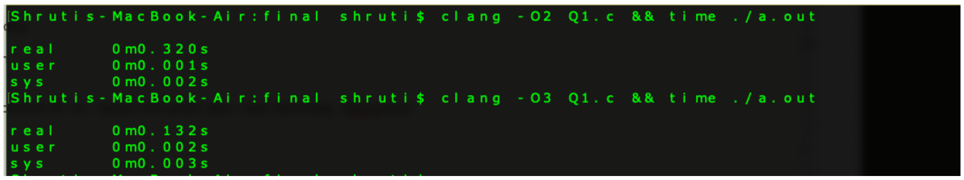

Figure 5: Time usage

2.3.2 Example 2 (Q3-2.c)

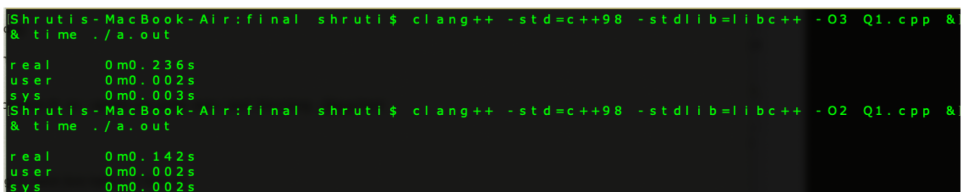

Figure 6: Time usage

The O3 option is recommended for applications that have loops that heavily use floating-point calculations and process large data sets. Loop optimizations like auto-vectorization is enabled at -O3

3 Task 3

Clang duplicates the type conversion for each use of the variable when pointers are involved.

LLVM does not currently support different rounding modes. Similarly, it does not yet support access to the floating-point environment, which makes reliable checks for floating-point exceptions in clang impossible.