



中山大學
SUN YAT-SEN UNIVERSITY

计算机学院（软件学院）

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Compiler Design

编译器构造实验

Lab 14: Project-4

张献伟

xianweiz.github.io

DCS292, 5/23/2023

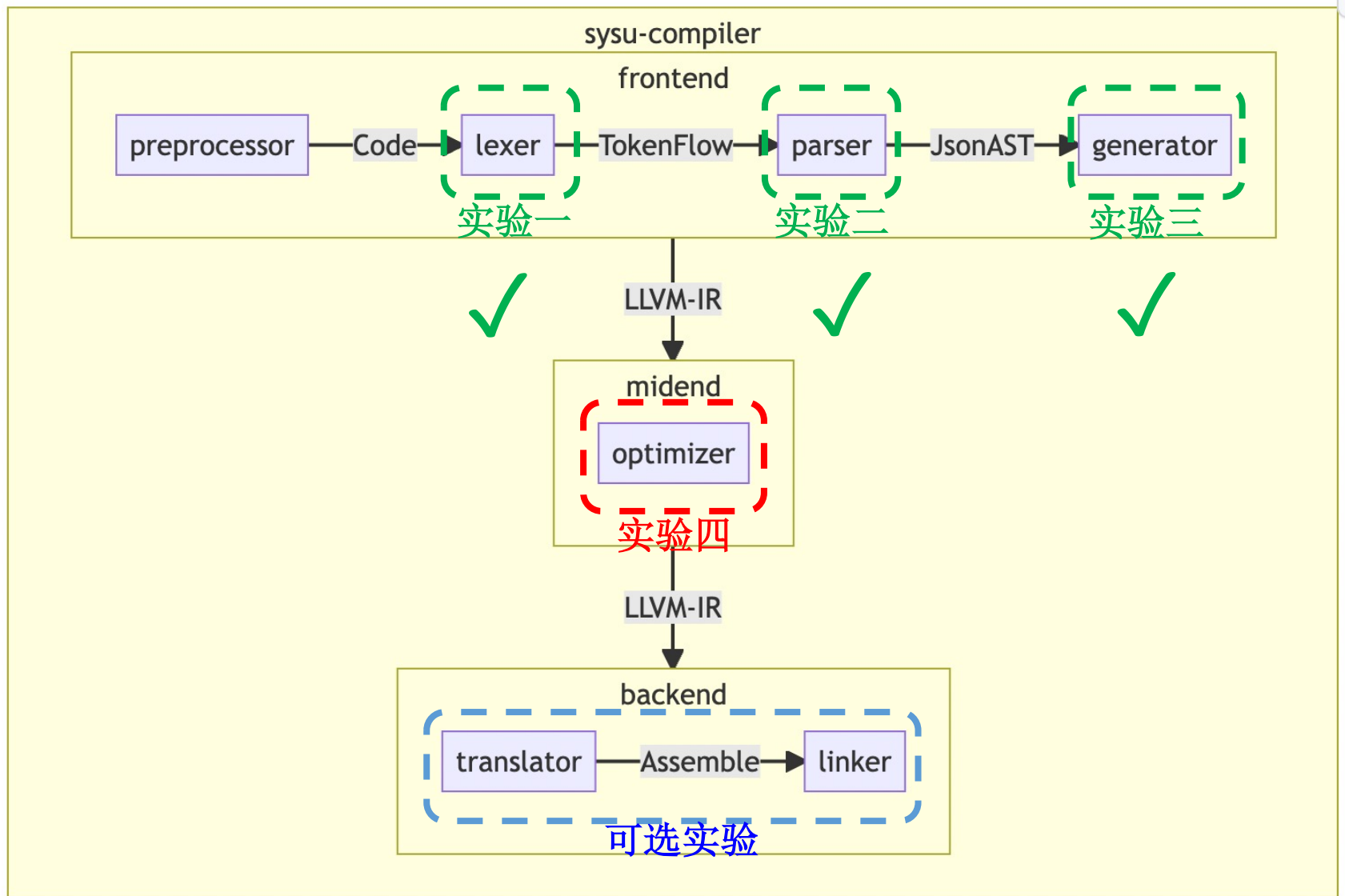
Project 4: What?

- 文档描述：
 - Readme: <https://github.com/arcsysu/SYsU-lang/tree/latest/optimizer>
 - Wiki: <https://github.com/arcsysu/SYsU-lang/wiki/实验三代码优化>
- 实现一个IR优化器
 - 输入：LLVM-IR（由Project 3或Clang提供）
 - 输出：LLVM-IR（优化版本）
- 总体流程
 - 引入Project3的IR（或使用clang）
 - 写analysis和transform passes
- 截止时间
 - **6/13/2023**

Project 4: How?

- 实现
 - `$vim optimizer/optimizer.cc`
- 编译
 - `$cmake --build ~/sysu/build -t install`
 - 输出: `~/sysu/build/optimizer/`
- 运行
 - `(export PATH=~/sysu/bin:$PATH \ CPATH=~/sysu/include:$CPATH \ LIBRARY_PATH=~/sysu/lib:$LIBRARY_PATH \ LD_LIBRARY_PATH=~/sysu/lib:$LD_LIBRARY_PATH && clang -E ../tester/functional/000_main.sysu.c | <THE_IR> | sysu-optimizer)`
 - Clang提供IR: `<THE_IR> = clang -cc1 -O0 -S -emit-llvm`
 - Project3提供IR: `<THE_IR > = sysu-generator`
 - `sysu-optimizer`
 - `opt -S --enable-new-pm -load-pass-plugin= libsysuOptimizer.so -passes="sysu-optimizer-pass"`

Schedule[实验安排]



LLVM Opt.

```
int main() {  
    int a, b, c;  
    a = b + c;  
    a = 3;  
  
    if (a > 0) return 1;  
    else return 0;  
}
```

clang -emit-llvm -S -O0 xx.c

```
define dso_local i32 @main() #0 {  
    %1 = alloca i32, align 4  
    %2 = alloca i32, align 4  
    %3 = alloca i32, align 4  
    %4 = alloca i32, align 4  
    store i32 0, i32* %1, align 4  
    %5 = load i32, i32* %3, align 4  
    %6 = load i32, i32* %4, align 4  
    %7 = add nsw i32 %5, %6  
    store i32 %7, i32* %2, align 4  
    store i32 3, i32* %2, align 4  
    %8 = load i32, i32* %2, align 4  
    %9 = icmp sgt i32 %8, 0  
    br i1 %9, label %10, label %11  
  
10:  
    store i32 1, i32* %1, align 4  
    br label %12  
  
11:  
    store i32 0, i32* %1, align 4  
    br label %12  
  
12:  
    %13 = load i32, i32* %1, align 4  
    ret i32 %13  
}
```

clang -emit-llvm -S -O1 xx.c

```
define dso_local i32 @main() local_unnamed_addr #0 {  
    ret i32 1  
}
```

LLVM Pass - Analysis vs Transformation

- A pass operates on some unit of IR (e.g. Module or Function)
 - Transformation pass will modify it
 - Analysis pass will generate some high-level information
- Analysis results are produced lazily
 - Another pass needs to request the results first
 - Results are cached
 - Analysis manager deals with a non-trivial cache (in)validation problem
- Transformation pass managers (e.g. FunctionPassManager) record what's preserved
 - Function pass can invalidate Module analysis results, and vice-versa

LLVM Pass

- `$clang -emit-llvm -S sum.c`

```
int sum(int a, int b) {  
    return a + b;  
}
```

- `$opt sum.ll -debug-pass=Structure -mem2reg -S -o sum-O1.ll`

Pass Arguments: `-targetlibinfo -tti -targetpassconfig -assumption-cache-tracker -domtree -mem2reg -verify -print-module`

Target Library Information

Target Transform Information

Target Pass Configuration

Assumption Cache Tracker

ModulePass Manager

FunctionPass Manager

Dominator Tree Construction

Promote Memory to Register

Module Verifier

Print Module IR

```
$opt sum.ll -debug-pass=Structure -O1 -S -o sum-O1.ll
```

```
$opt sum.ll -time-passes -O1 -o sum-tim.ll
```

- `$opt sum.ll -time-passes -mem2reg -o sum-tim.ll`

```
=====  
... Pass execution timing report ...  
=====  
Total Execution Time: 0.0003 seconds (0.0003 wall clock)  
  
---User Time---  --System Time--  --User+System--  ---Wall Time---  --- Name ---  
0.0002 ( 91.1%)  0.0001 ( 90.2%)  0.0003 ( 90.8%)  0.0003 ( 90.6%)  Bitcode Writer  
0.0000 (  3.7%)  0.0000 (  4.5%)  0.0000 (  4.0%)  0.0000 (  3.7%)  Module Verifier  
0.0000 (  2.3%)  0.0000 (  2.3%)  0.0000 (  2.3%)  0.0000 (  2.8%)  Dominator Tree Construction  
0.0000 (  2.3%)  0.0000 (  2.3%)  0.0000 (  2.3%)  0.0000 (  2.4%)  Promote Memory to Register  
0.0000 (  0.5%)  0.0000 (  0.8%)  0.0000 (  0.6%)  0.0000 (  0.6%)  Assumption Cache Tracker  
0.0002 (100.0%)  0.0001 (100.0%)  0.0003 (100.0%)  0.0003 (100.0%)  Total  
  
=====  
LLVM IR Parsing  
=====  
Total Execution Time: 0.0006 seconds (0.0006 wall clock)
```

main.cc

```
38 // Create a module pass manager and add StaticCallCounterPrinter to it.
39 llvm::ModulePassManager MPM; https://llvm.org/doxygen/classllvm\_1\_1PassManager.html
40 MPM.addPass(sysu::StaticCallCounterPrinter(llvm::errs()));
41
42 // Create an analysis manager and register StaticCallCounter with it.
43 llvm::ModuleAnalysisManager MAM; https://llvm.org/doxygen/classllvm\_1\_1AnalysisManager.html
44 MAM.registerPass([& { return sysu::StaticCallCounter(); }]);
45
46 // Register all available module analysis passes defined in PassRegistry.def.
47 // We only really need PassInstrumentationAnalysis (which is pulled by
48 // default by PassBuilder), but to keep this concise, let PassBuilder do all
49 // the _heavy-lifting_.
50 llvm::PassBuilder PB; 注册所有的分析Pass
51 PB.registerModuleAnalyses(MAM);
52
53 // Finally, run the passes registered with MPM
54 MPM.run(*M, MAM); M: the IR module MAM: the analysis manager
55
56 M->print(llvm::outs(), nullptr);
```

```
void PassBuilder::registerModuleAnalyses(
    ModuleAnalysisManager &MAM) {
    #define MODULE_ANALYSIS(NAME, CREATE_PASS)
    MAM.registerPass([& { return CREATE_PASS; }]);
    #include "PassRegistry.def"

    e.g. PB.registerAnalysisRegistrationCallback
    for (auto &C : ModuleAnalysisRegistrationCallbacks)
        C(MAM);
}
```

PassBuilder.cpp

StaticCallCounterPrinter

optimizer.hh

```
27 class StaticCallCounterPrinter
28     : public llvm::PassInfoMixin<StaticCallCounterPrinter> {
29 public:
30     explicit StaticCallCounterPrinter(llvm::raw_ostream &OutS) : OS(OutS) {}
31     llvm::PreservedAnalyses run(llvm::Module &M, 声明run()方法, which actually runs the pass
32                                 llvm::ModuleAnalysisManager &MAM);
33
34 private:
35     llvm::raw_ostream &OS;
36 };
```

run(): 接收一些IR单元和一个分析管理器, 返回类型为 PreservedAnalyses

LLVM:

```
template <typename DerivedT> struct PassInfoMixin {
    static StringRef name() {
        // (...)
    }
};

template <typename IRUnitT,
          typename AnalysisManagerT = AnalysisManager<IRUnitT>,
          typename... ExtraArgTs>
class PassManager : public PassInfoMixin<
    PassManager<IRUnitT, AnalysisManagerT, ExtraArgTs...>> {

    PreservedAnalyses run(IRUnitT &IR, AnalysisManagerT &AM,
        ExtraArgTs... ExtraArgs) {
        // Passes is a vector of PassModel<> : PassConcept
        for (unsigned Idx = 0, Size = Passes.size(); Idx != Size; ++Idx) {
            PreservedAnalyses PassPA = P->run(IR, AM, ExtraArgs...);

            AM.invalidate(IR, PassPA);
        }
    } // end of run
} // end of PassManager
```

llvm/include/llvm/IR/PassManager.h

StaticCallCounterPrinter (cont.)

optimizer.cc

```
5  llvm::PreservedAnalyses
6  sysu::StaticCallCounterPrinter::run(llvm::Module &M, 实现run()方法
7                                     llvm::ModuleAnalysisManager &MAM) {
8
9      auto DirectCalls = MAM.getResult<sysu::StaticCallCounter>(M);
10
11     OS << "=====\n";
12     OS << "sysu-optimizer: static analysis results\n";
13     OS << "=====\n";
14     const char *str1 = "NAME", *str2 = "#N DIRECT CALLS";
15     OS << llvm::format("%-20s %-10s\n", str1, str2);
16     OS << "-----\n";
17
18     for (auto &CallCount : DirectCalls) {
19         OS << llvm::format("%-20s %-10lu\n",
20                             CallCount.first->getName().str().c_str(),
21                             CallCount.second);
22     }
23
24     OS << "-----\n\n";
25     return llvm::PreservedAnalyses::all();
26 }
```

Get the result of an analysis pass for a given IR unit

你可以：
获取分析Pass的结果，然而优化修改代码

StaticCallCounter

optimizer.hh

```
15 class StaticCallCounter : public llvm::AnalysisInfoMixin<StaticCallCounter> {
16 public:
17     using Result = llvm::MapVector<const llvm::Function *, unsigned>;
18     Result run(llvm::Module &M, llvm::ModuleAnalysisManager &);
19
20 private:
21     // A special type used by analysis passes to provide an address that
22     // identifies that particular analysis pass type.
23     static llvm::AnalysisKey Key;
24     friend struct llvm::AnalysisInfoMixin<StaticCallCounter>;
25 };
```

```
struct AnalysisInfoMixin : PassInfoMixin<DerivedT> {
    /// Returns an opaque, unique ID for this analysis type.
    ///
    static AnalysisKey *ID() {
        static_assert(std::is_base_of<AnalysisInfoMixin, DerivedT>::value,
            "Must pass the derived type as the template argument!");
        return &DerivedT::Key;
    }
};
```

[llvm/include/llvm/IR/PassManager.h](#)

StaticCallCounter (cont.)

optimizer.cc

```
28 sysu::StaticCallCounter::Result
29 sysu::StaticCallCounter::run(llvm::Module &M, llvm::ModuleAnalysisManager &) {
30     llvm::MapVector<const llvm::Function *, unsigned> Res;
31
32     for (auto &Func : M) {
33         for (auto &BB : Func) {
34             for (auto &Ins : BB) {
35
36                 // If this is a call instruction then CB will be not null.
37                 auto *CB = llvm::dyn_cast<llvm::CallBase>(&Ins);
38                 if (nullptr == CB) {
39                     continue;
40                 }
41
42                 // If CB is a direct function call then DirectInvoc will be not null.
43                 auto DirectInvoc = CB->getCalledFunction();
44                 if (nullptr == DirectInvoc) {
45                     continue;
46                 }
47
48                 // We have a direct function call - update the count for the function
49                 // being called.
50                 auto CallCount = Res.find(DirectInvoc);
51                 if (Res.end() == CallCount) {
52                     CallCount = Res.insert({DirectInvoc, 0}).first;
53                 }
54                 ++CallCount->second;
55             }
56         }
57     }
58
59     return Res;
60 }
```



Pass Registration

optimizer.cc

```
65  llvm::PassPluginLibraryInfo LLVM_ATTRIBUTE_WEAK llvmGetPassPluginInfo() {
66      return {LLVM_PLUGIN_API_VERSION, "sysu-optimizer-pass", LLVM_VERSION_STRING,
67              [](llvm::PassBuilder &PB) {
68                  // #1 REGISTRATION FOR "opt -passes=sysu-optimizer-pass"
69                  PB.registerPipelineParsingCallback(
70                      [&](llvm::StringRef Name, llvm::ModulePassManager &MPM,
71                          llvm::ArrayRef<llvm::PassBuilder::PipelineElement>) {
72                          if (Name == "sysu-optimizer-pass") {
73                              MPM.addPass(sysu::StaticCallCounterPrinter(llvm::errs()));
74                              return true;
75                          }
76                          return false;
77                      });
78                  // #2 REGISTRATION FOR
79                  // "MAM.getResult<sysu::StaticCallCounter>(Module)"
80                  PB.registerAnalysisRegistrationCallback(
81                      [](llvm::ModuleAnalysisManager &MAM) {
82                          MAM.registerPass([&] { return sysu::StaticCallCounter(); });
83                      });
84      }};
85 }
```

LLVM:

```
struct PassPluginLibraryInfo {
    // The API version understood by this plugin
    uint32_t APIVersion;
    // A meaningful name of the plugin.
    const char *PluginName;
    // The version of the plugin.
    const char *PluginVersion;

    // Callback for registering plugin passes with PassBuilder
    void (*RegisterPassBuilderCallbacks)(PassBuilder &);
};
```

[include/llvm/Passes/PassPlugin.h](#)

```
// Load requested pass plugins and let them register pass
// builder callbacks
bool runPassPipeline(...) {
    from CL option
    for (auto &PluginFN : PassPlugins) {
        auto PassPlugin = PassPlugin::Load(PluginFN);
        FPMHook from HelloWorld.cpp
        PassPlugin->registerPassBuilderCallbacks(PB);
    }
}
```

[tools/opt/NewPMDriver.cpp](#)

References

- The New Pass Manager
 - <https://blog.llvm.org/posts/2021-03-26-the-new-pass-manager/>
- Using the New Pass Manager
 - <https://llvm.org/docs/NewPassManager.html>
- Writing an LLVM Pass: 101 (LLVM 2019 tutorial)
 - <https://llvm.org/devmtg/2019-10/slides/Warzynski-WritingAnLLVMPass.pdf>
- Writing an LLVM Pass
 - <https://llvm.org/docs/WritingAnLLVMNewPMPass.html>
- LLVM's Analysis and Transform Passes
 - <https://www.llvm.org/docs/Passes.html>
- Getting Started with LLVM Core Libraries
 - <https://faculty.sist.shanghaitech.edu.cn/faculty/songfu/course/spring2018/CS131/llvm.pdf>