Advanced Topics in Compilers

Simone Campanoni
simone.campanoni@northwestern.edu
Outline

• Introducing NOELLE

• Building upon NOELLE

• Documentation
Software framework: NOELLE

• Git repo: https://github.com/arcana-lab/noelle

• You need to use LLVM 9.0.0
  • On hanlon.wot.eecs.northwestern.edu:
    LLVM_HOME= /home/software/llvm-9.0.0
    export PATH=$LLVM_HOME/bin:$PATH ;
    export LD_LIBRARY_PATH=$LLVM_HOME/lib:$LD_LIBRARY_PATH
  • On peroni.cs.northwestern.edu
    source /project/extra/llvm/9.0.0/enable

• Try to compile the framework
  $ git clone https://github.com/arcana-lab/noelle
  $ cd noelle
  $ make
Software framework: NOELLE

• Problem:
  • LLVM provides low-level and only code-centric APIs to middle-end passes
  • This makes the design of advanced code analyses and transformations hard

• Solution:
  • NOELLE complements LLVM by providing a dependence-centric (and more expensive, unfortunately) APIs at different granularities to middle-end passes
  • Even advanced code transformations (code parallelization, code vectorization, loop transformations) can be now implemented in a few lines of code (less than 1000!!)
  • NOELLE’s APIs are optional and you can combine them with LLVM’s APIs
  • For most NOELLE’s APIs:
    • You pay the cost of an API provided by NOELLE when you invoke that API
Current limitations of NOELLE

• You can analyze and/or transform a program, but not a library
  • The existence of `main` is assumed

• The IR code being analyzed/transformed using NOELLE has to be (at least) normalized using `noelle-norm`

• You keep track of which abstractions are no longer valid due to changes you have made to the code
  • Suggestion: use all abstractions you need to decide what to do, then do all changes at once
  • Suggestion: you can invoke your NOELLE-based transformation until a fixed-point is reached (learn how to use `noelle-fixedpoint`)
Compiling and installing NOELLE

• NOELLE is configured to be compiled with SCAF, SVF, and LLVM alias analyses by default

• The installation directory is (by default) the sub-directory ./install of the NOELLE repository

• If you want to change NOELLE’s default configuration, please run: make menuconfig

• To compile and install NOELLE, run from the root directory of the repository: make
NOELLE structure

- Examples of LLVM middle-end passes built upon NOELLE
- NOELLE’s internals
- NOELLE’s tests
  - Unit tests

After you compile NOELLE,
NOELLE’s
- Binaries
- public APIs
- tools
NOELLE structure

Abstractions provided by NOELLE and their public APIs

All of them are within the namespace arcana::noelle

Tools/analyses built upon NOELLE
NOELLE structure

Simple examples of LLVM passes that use NOELLE’s abstractions/APIs

Simple C/C++ programs that can be used to test the simple LLVM passes built using NOELLE
NOELLE commands

Code normalizations:

• noelle-norm
  normalize LLVM IR for NOELLE

• noelle-simplification
  run simple transformations that remove some redundancy (e.g., constant propagation)
  and then normalize LLVM IR for NOELLE
NOELLE commands

Metadata:

• noelle-meta-X-clean
  Remove metadata related to X from the given LLVM IR file

• noelle-meta-X-embed
  Embed metadata related to X into the given LLVM IR file

• The X can be:
  • loop  about IDs of single loops
  • pdg   about the memory dependences needed to build a PDG
  • prof  about data generated by profilers
  • scc   about the composition of single SCCs
NOELLE commands

Miscellaneous:

- noelle-prof-coverage
  Generate a binary with profiling instructions from a given LLVM IR file

- noelle-pdg
  Print the PDG of a given LLVM IR file

- noelle-load
  Replacement for opt to be used to run an LLVM pass built upon NOELLE (see next)
Outline

• Introducing NOELLE

• Building upon NOELLE

• Documentation
Middle-end pass

• Download the skeleton for an LLVM middle-end pass from here: https://github.com/scampanoni/LLVM_middleend_template

• You need to (see next slides):
  1. Extend/change this skeleton (src/CatPass.cpp) to implement your middle-end pass upon NOELLE
  2. Declare NOELLE to cmake (src/CMakeLists.txt)
Middle-end pass

1. Extend/change this skeleton (src/CatPass.cpp) to implement your middle-end pass upon NOELLE
namespace {
    struct CAT : public FunctionPass {
        static char ID;

        CAT() : FunctionPass(ID) {}  

        bool doInitialization (Module &M) override {
            errs() << "Hello" << __FILE__ << __LINE__ << std::endl;
            return false;
        }

        bool runOnFunction (Function &F) {
            errs() << "Hello" << __FILE__ << __LINE__ << std::endl;
            return false;
        }

        void getAnalysisUsage (AnalyzerContext &A) {
            errs() << "Hello" << __FILE__ << __LINE__ << std::endl;
            AU.setPreservesCFG();
        }
    }
}

#include "llvm/Pass.h"
#include "llvm/IR/Function.h"
#include "llvm/Support/raw_ostream.h"
#include "llvm/IR/LegacyPassManager.h"
#include "llvm/Transforms/IpO/PassManagerBuilder.h"

using namespace llvm;
CatPass.cpp using NOELLE

```cpp
#include "llvm/Pass.h"
#include "llvm/IR/Function.h"
#include "llvm/Support/raw_ostream.h"
#include "llvm/IR/LegacyPassManager.h"
#include "llvm/Transforms/IPO/PassManagerBuilder.h"
#include "noelle/core/Noelle.hpp"

using namespace arcana::noelle;

struct CAT : public ModulePass {
  static char ID;

  CAT() : ModulePass(ID) {}

  bool doInitialization (Module &M) override {
    return false;
  }

  bool runOnModule (Module &M) override {
    /*
    * Fetch NOELLE
    */
    auto& noelle = getAnalysis<Noelle>();

    /*
    * Use NOELLE
    */
    auto insts = noelle.numberOfProgramInstructions();
    errs() << "The program has " << insts << " instructions\n";
    return false;
  }

  void getAnalysisUsage(AnalysisUsage &AU) const override {
    AU.addRequired<Noelle>();
  }
};
```

It has to be a ModulePass

Fetch NOELLE

Simple example of using NOELLE

Declare to LLVM that your pass depends on NOELLE
Middle-end pass

1. Extend/change this skeleton (src/CatPass.cpp) to implement your middle-end pass upon NOELLE
2. Declare NOELLE to cmake (src/CMakeLists.txt)
Declaring NOELLE to cmake

• Put NOELLE in your environment:
  source MY_NOELLE/enable

• Find out where is the include directory of your installed NOELLE
  noelle-config --include

• Copy the directory printed above (e.g., MY_NOELLE/install/include)
  and paste it into src/CMakeLists.txt
  include_directories(${LLVM_INCLUDE_DIRS} MY_NOELLE/install/include)

• You can now compile your pass built upon NOELLE.
  To do so, run the following script from your LLVM pass root directory:
  ./run_me.sh
Running NOELLE based passes

• noelle-load rather than opt

• In **CS 323**:  
  • opt –load ~/CAT/lib/CAT.so -CAT A.bc –o B.bc

• Now:
  • noelle-load –load ~/CAT/lib/CAT.so -CAT A.bc –o B.bc

It will print the invocation to opt with all arguments if you invoke it with “--noelle-verbose=1” (or 2, 3)

```bash
opt -load /nfs-scratch/simonec/parallelism/parallelization/NOELLEs/2/install/lib/CallGraph.so ...
-load /home/simonec/CAT/lib/MYPASS.so -MYPASS A.bc -o B.bc
```
Let’s compile a simple example of code transformation built upon NOELLE

• cd examples/passes
• make links ; cd simple
• ./scripts/run_me.sh

It will compile and install the pass to ~/CAT (like in 323)
Let’s run a simple example of code transformation built upon NOELLE

• cd examples/tests

• source ../../enable ;

• cd 0 ;

• make -f Makefile_no_profile

You have to normalize the code before invoking NOELLE
Outline

• Introducing NOELLE

• Building upon NOELLE

• Documentation
Documentation of NOELLE

• Entry point: README.md

• All links to other documentation/videos/slides are reachable from the entry point

• Please read the documentation (most questions can be answered by reading the documentation)
Always have faith in your ability

Success will come your way eventually

Best of luck!