Advanced Topics in Compilers

NOELLE

Simone Campanoni
simone.campanoni@northwestern.edu
Outline

• NOELLE’s code structure

• Building upon NOELLE

• Developing NOELLE
Software framework: NOELLE

• Git repo: https://github.com/scampanoni/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/scampanoni/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 / transform a program, but not a library
  • The existence of main is assumed
  • The whole program is assumed

• The IR code being analyzed/transformed using NOELLE is (at least) normalized using noelle-norm

• You keep track of which abstractions are not 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 NOELLE multiple times (learn how to use noelle-fixedpoint)
NOELLE structure

Examples of LLVM middle-end passes built upon NOELLE

NOELLE’s internals

NOELLE’s tests
  • Unit tests
  • Regression tests
  • Performance tests

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

Abstractions provided by NOELLE and their public APIs

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
Outline

- NOELLE’s code structure
- Building upon NOELLE
- Developing NOELLE
namespace {
    struct CAT : public FunctionPass {
        static char ID;

        CAT() : FunctionPass(ID) {} // Next there is code to register your pass to "opt"

        bool doInitialization (Module &M) override {
            errs() << "Hello AWB\n            return false;
            char CAT::ID = 0;

            static RegisterPass<CAT> X("CAT", "Homework for the CAT class");

            bool runOnFunction (Function &F) override {
                errs() << "Hello AWB"
                return false;

            }

            void getAnalysisUsage (AnalysisUsage &AU) const {
                AU.setPreservesCFG();
            }
        } // Next there is code to register your pass to "clang"

        static CAT * _PassMaker = NULL;

        static RegisterStandardPasses _RegPass1(PassManagerBuilder::EP_OptimizerLast,  // ** for -O0
            [](const PassManagerBuilder&, legacy::PassManagerBase& PM) {
                if(!_PassMaker){ PM.add(_PassMaker = new CAT());}});

        static RegisterStandardPasses _RegPass2(PassManagerBuilder::EP_OptimizerOn,  // ** for -O0
            [](const PassManagerBuilder&, legacy::PassManagerBase& PM) {
                if(!_PassMaker){ PM.add(_PassMaker = new CAT());}});
    }
}

```cpp
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>();
    }
};
```

Declare to LLVM that your pass depends on NOELLE

Fetch NOELLE

Simple example of using NOELLE
Running NOELLE based passes

• noelle-load **rather than** opt

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

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

It will print the invocation to opt with all arguments (in case it will debugging)

```
opt -load /nfs-scratch/simonec/parallelism/parallelization/NOELLEs/2/install/lib/CallGraph.so
    -load /nfs-scratch/simonec/parallelism/parallelization/NOELLEs/2/install/lib/libSvf.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

• NOELLE’s code structure

• Building upon NOELLE

• Developing NOELLE
Developing and testing

• Let’s say you are working to improve a NOELLE’s module (e.g., induction variable detection algorithm)
  
  • You need to test the correctness and impacts of your work. …
  
  • NOELLE can help you do that
Testing

• NOELLE includes tests for its code transformations (e.g., code parallelization, loop-invariant code motion, etc...)
Testing

- NOELLE includes tests for its code transformations (e.g., code parallelization, loop-invariant code motion, etc...)

```
Contributing.md LICENSE.md Makefile README.md
doc examples external src tests
```

- `cd tests ;`
- If you don’t have condor installed in your platform
- `make`
- It runs the transformations only using their default configurations (e.g., unroll-factor set to be the default one)
- `make condor`
- If you have condor installed in your platform
- It generates condor files to run in parallel all transformations with many different configurations (generating more than 20,000 tests that all run in parallel)
Testing with condor

cd tests ; make condor

- Contributing.md
- LICENSE.md
- Makefile
- README.md
- doc
- examples
- external
- src
- tests

Makefile

- condor
- performance
- regression
- scripts
- unit

... regression_65
... regression_66
... regression_67
... regression_68
... regression_69
... regression_7
... regression_70
... regression_71
... regression_72
... regression_73
... regression_74
... regression_75
... regression_76
... regression_77
... ...

Copy of the original regression dir one directory per configuration for the code transformations.

All these tests (~20,000 at the moment) run in parallel!
Testing with condor

```bash
cd tests ; make condor

Makefile
  condor
  performance
  regression
  scripts
  unit

... regression_65
regression_66
regression_67
regression_68
regression_69
regression_7
regression_70
regression_71
regression_72
regression_73
regression_74
regression_75
regression_76
regression_77
...
```

```bash
cd tests ; make condor_check

$ make condor_check
./scripts/condor_check.sh:

# REGRESSION TESTS:
Checking the regression test results
There are 21204 jobs that are still running
No new tests failed so far
There are new tests that now pass for all configurations. They are the next ones:

  Chunking
  DSWPIterations_RemovalIntraIterMemEdge
  Exit_call2
  Exit_call3
  IndependentIterations11
  IndependentIterations5
  LION
  LION_2
  Multiloops
  Multiloops_list
  ReductionIterationsAnd
  ReductionIterationsOr

# UNIT TESTS:
They are still running

# PERFORMANCE TESTS:
They are still running
```
Testing with condor

cd tests ; make condor

Makefile
condor
performance
regression
scripts
unit

regression_65
regression_66
regression_67
regression_68
regression_69
regression_7
regression_70
regression_71
regression_72
regression_73
regression_74
regression_75
regression_76
regression_77

• Tests that completed successfully get automatically deleted
• Directory of a test that failed is kept (so you can debug it; check \texttt{compiler_output.txt}) and a script to reproduce the fail is automatically generated
• To reproduce the fail:
  • Go to the directory of the test (e.g., \texttt{cd regression\_4/Simple})
  • Run \texttt{./run\_me.sh}
Re-run the tests using condor

cd tests;

1. Make sure no tests are still running
   condor_q `whoami`

2. Clean the tests directory
   make clean

3. Run the tests
   make condor
Running a single test without condor

cd tests ; make download

1. Go to the test directory (e.g., cd regression/Simple)
2. Clean the directory
   make clean
3. Enable NOELLE binaries in your environment
   source ../..../..enable
4. Run the test
   make test_correctness
5. Check the output
   (look at the makefile to understand the scripts)
Always have faith in your ability

Success will come your way eventually

Best of luck!