Using LLVM

Daya S Khudia
(uniqname: dskhudia)

Office Hours: 3-5 in 1695 CSE
Tues, Thurs and Friday

What is LLVM?

- Modular and reusable compiler technologies.
- Supports a variety of front-ends (C/C++/Fortran).
- Code can be generated for variety of targets (ARM/Alpha/X86).

- Pass framework is an important part of it
  - Modular and well separated from core compiler parts
  - Many transformations and optimization can be written as passes on intermediate representation (IR).
  - IR is also called llvm-bitcode.

Running a sample program

- llvm-gcc is installed on andrew, hugo and wilma
- To use it add it to your path
  - csh/tcsh users
    - *setenv PATH /y/583f11/llvm-gcc-install/bin:$PATH*
  - bash users
    - *export PATH=/y/583f11/llvm-gcc-install/bin:$PATH*
- Assuming LLVM tools (e.g. opt, llc) are already in your path
  - *source file: knapsack.c*
    - *llvm-gcc -emit-llvm knapsack.c -c -o knapsack.bc*
    - *llc knapsack.profile.bc -o knapsack.profile.s*
    - *g++ -o knapsack knapsack.s*
    - *./knapsack*

Using profiling

- Inserting Profiling
  - *llvm-gcc -emit-llvm knapsack.c -c -o knapsack.bc*
  - *opt -insert-edge-profiling knapsack.bc -o knapsack.profile.bc*
  - *llc knapsack.profile.bc -o knapsack.profile.s*
  - *g++ -o knapsack.profile knapsack.profile.s <LLVM_OBJ_ROOT>/<Debug|Release|Debug+Asserts>/lib/profile_rt.so*
  - *./knapsack.profile*
- *The previous step will generate llvmprof.out. You can load it using following command while running your pass (hw1pass is the pass name. You can use any name)*
  - *opt -load <PASS_OBJ_ROOT>/Debug+Asserts/lib/hw1pass.so -profile-loader -profile-info-file=llvmprof.out -hw1pass knapsack.bc >& /dev/null*

Sample Project

- Keep your code separate from LLVM
  - Copy the <LLVM_SRC_ROOT>/projects/sample to a new directory
  - Change following variables as indicated
    - AC_INIT([[[hw1pass]]],[[[1.00]]],[bugs@yourdomain])
    - LLVM_SRC_ROOT="<your_llvm_source_path>"
    - LLVM_OBJ_ROOT="<your_llvm_build_dir>"
    - AC_CONFIG_MAKEFILE(lib/hw1pass/Makefile)
    - AC_CONFIG_MAKEFILE(tools/hw1pass/Makefile)
    - add "LOADABLE_MODULE=1" in hw1pass/Makefile for making a shared library
    - change the file names and dir names in make files (if you change sample.[ch] to hw1pass.[ch])
    - in autoconf dir run theand provide llvm src/obj paths
      - ./AutoRegen.sh
  - Now you can configure and build your pass in the same or different directory.

Debugging Tools

- Viewing the bitcode
  - knapsack.bc generating by llvm-gcc is binary file
  - To view the bitcode, run
    - *llvm-dis knapsack.bc*
  - It will generate knapsack.ll and you can view that using an editor of your choice.
- Generating the CFG. The following command will dump a separate dot file for each function in the knapsack.c program
  - *opt -dot-cfg knapsack.bc >& /dev/null*
- To view the CFG, you can use following tools (installed on hurricane machines)
  - xdot <cfg*.dot file>
  - dotty <cfg*.dot file>

Running GDB

- Running gdb is tricky because the passes are dynamically loaded
  - To run gdb, you need to have a
    - debug build of LLVM installed
    - debug build of your pass
- Refer to
  - http://llvm.org/docs/WritingAnLLVMPass.html#debughints
  - The commands are
    - gdb <LLVM_INSTALL_DIR>/bin/opt
    - break llvm::PassManager::run
    - run <options>
  - After running above commands you will be able to set breakpoints on the functions written in your pass
    - Take care of anonymous namespace if you are using it

Interesting components of a Pass

```
#include "llvm/Pass.h"
#include "llvm/Function.h"
#include "llvm/Support/raw_ostream.h"

using namespace llvm;
namespace {
    struct statComp: public FunctionPass {
        static char ID;
        ProfileInfo* PI;
        statComp() : FunctionPass(ID) {}
        virtual bool runOnFunction(Function &F) {
            PI = &getAnalysis<ProfileInfo>();
            double intAddCount = 0;
            for(Function::iterator b = F.begin(), be = F.end(); b != be; ++b) {
                for(BasicBlock::iterator i = b->begin(), ie = b->end(); i != ie; i++){
                    if(i->getOpcode == 8){
                        intAddCount += PI->getExecutionCount(b);
                    }
                }
            }
            return false; //return true if you modified the code
        }
    };
}
```

Note: The pass itself is incomplete and it might not compile as it is.

LAMP Profiling

- Download the HW1LAMP.tgz from the course website
  - Read README
  - The commands to get memory profiling data and to load it are in the README file.
  - Make sure the result.lamp.profile file is in the same dir where you are running opt command.
- Extracting information from LAMP data structures (Look into LAMP/LAMPLoadProfile.h)

```
std::map<unsigned int, Instruction*> IdToInstMap;   // Inst* -> InstId
std::map<Instruction*, unsigned int> InstToIdMap;   // InstID -> Inst*
std::map<std::pair<Instruction*, Instruction*>*, unsigned int> DepToTimesMap;
```