Project 0

Advanced Compilers

Released: April 29, 2025
Due: May 12, 2025

In this project, we will setup LLVM environment and plot control flow graph (CFG) using LLVM. Since changing default compiler executable or option can affect whole system, it's always good to separate your base environment from experimental settings. We will use docker container for such isolation.

1 Environment setups

1.1 Docker

Please refer to the official docker's installation guide at https://docs.docker.com/engine/install. One can add a user into 'docker' group to grant permissions.

```
sudo usermod -aG docker $(whoami)
```

Otherwise, use docker commands after 'sudo'. One can check running containers by

```
docker ps
```

Also, one can open the bash shell on the container via

```
docker exec -it <conatiner_name> bash
```

1.2 Install LLVM

TL;DR : Run '.container/deploy.sh'.
Former script will launch a container named '`advcmp`' and do most of tedious setups for you, and let you jump to section 2. If multiple people have to do project on the same server, please configure '`container_name`' at '`deploy.sh`'. The script will register repository and install llvm toolchain for you.

1.2.1 Using package manager (Ubuntu)

LLVM officially supports apt repository. https://apt.llvm.org/

- Add following repository links at '/etc/apt/sources.list.d/llvm.list'.

```
# latest (e.g. ubuntu 24.04: <ubuntu_version>=noble)
deb http://apt.llvm.org/<ubuntu_version>/ llvm-toolchain-<
    ubuntu_version> main
deb-src http://apt.llvm.org/<ubuntu_versoin>/ llvm-toolchain-<
    ubuntu_version> main
```

- Get repository's signature.

```
wget -qO- https://apt.llvm.org/llvm-snapshot.gpg.key | sudo tee \
    /etc/apt/trusted.gpg.d/apt.llvm.org.asc
```

- Update and install.

```
# <N> is LLVM version (e.g. <N>=19)
sudo apt update
sudo apt install -y libllvm<N> llvm-<N> llvm-<N>-dev clang-<N> ...
```

  Installing '`libllvm<N>`', '`llvm-<N>`', and '`llvm-<N>-dev`' is mandatory. Also, we recommend you to install any version of '`lld`', '`lldb`', or '`clang`'.

- Test

```
llvm-config-<N> --help
```

- (Optional) Symbolic links
  One has to append version number to call installed `LLVM` or `clang` compiler. (e.g., `llvm-config-<N>`, `clang-<N>`) By using '`update-alternatives`' in Ubuntu, symbolic links without tailing version number can be easily generated.
  Please run the script '`.container/script/update-alternatives-llvm.sh`' **inside** the container. If you used given script ('`.container/deploy.sh`') to launch a container, you don't need to run the script manually.

1.2.2 Build from source

You will need '`cmake`', '`ninja-build`' and C/C++ compiler for bootstrapping build. Also, we recommend using '`ccache`' and '`lld`' to speed up build. For more details, please refer to https://llvm.org/docs/CMake.html

- Fetch source

```
git clone https://github.com/llvm/llvm-project.git
git checkout -t llvmorg-<N>
```

- Configure cmake

```
mkdir llvm-project/build
cd llvm-project/build
```

```
cmake -G Ninja ../llvm \
    -DCMAKE_CXX_STANDARD=17 \
    -DCMAKE_C_COMPILER=clang \
    -DCMAKE_CXX_COMPILER=clang++ \
    -DCMAKE_BUILD_TYPE=Release \
    -DCMAKE_INSTALL_PREFIX="/tmp/llvm" \
    -DLLVM_ENABLE_RTTI=ON \
    -DLLVM_TARGETS_TO_BUILD="host" \
    -DLLVM_ENABLE_ASSERTIONS=ON \
    -DLLVM_BUILD_LLVM_DYLIB=ON \
    -DLLVM_DYLIB_COMPONENTS="all" \
    -DLLVM_USE_LINKER="lld" \
    -DLLVM_CCACHE_BUILD=ON
```

  You may change install directory by changing '`-DCMAKE_INSTALL_PREFIX`', and compiler used by '`-DCMAKE_C_COMPILER`' and '`-DCMAKE_CXX_COMPILER`' options.

- Build and install

```
ninja
ninja install
```

- Configure library path

```
echo "/tmp/llvm/lib" | sudo tee /etc/ld.so.conf.d/llvm.conf
sudo ldconfig
```

2 Printing CFG

Workflow

1. Generate (emit) LLVM IR from the source.
2. Use '`dot-cfg`' pass via '`opt`'.
3. Convert generated `dot` file into image (`png`).

2.1 Emit LLVM IR

```
clang -S -emit-llvm -O0 -Xclang -disable-O0-optnone \
    -fno-discard-value-names <source> -o <out.ll>
```

- `-Xclang -disable-O0-optnone` :
  Tells compiler to not generate '`optnone`' attributes. '`optnone`' attribute prohibits `opt` to access and modify the given function or operation.
- `-fno-discard-value-names` :
  Make compiler to use value names written in the source code.
- The default file extension for LLVM IR is '`.ll`'.

2.2 Call `dot-cfg` pass

We will use '`opt`' to call pass over the generated LLVM IR.

```
opt -passes=dot-cfg <input.ll> -disable-output
```

`dot` files starting with '.(dot)' will be generated for each function in the IR.
Note that those files are hidden in default (because their name starts with '.(dot)'!). Please use '`ls`' with '`-a`' option to see those files are in place.

2.3 Convert '`dot`' into image

```
sudo apt install graphviz
dot -Tpng <dot_file> -o <output.png>
```

3 Submission

- Submit your CFG as '`PR0_<YOUR_STUDENT_ID>.png`' (e.g., '`PR0_2024-12345.png`') at eTL.

4 Example

```c
#include <stdio.h>

int main() {
    const int x = 12;
    const int y = 20;

    int i, j;
    for (j = 0; j < y; ++j) {
        for (i = 0; i < x; ++i) {
            if (i > j)
                continue;
            printf("*");
        }
        printf("\n");
    }
    return 0;
}
```

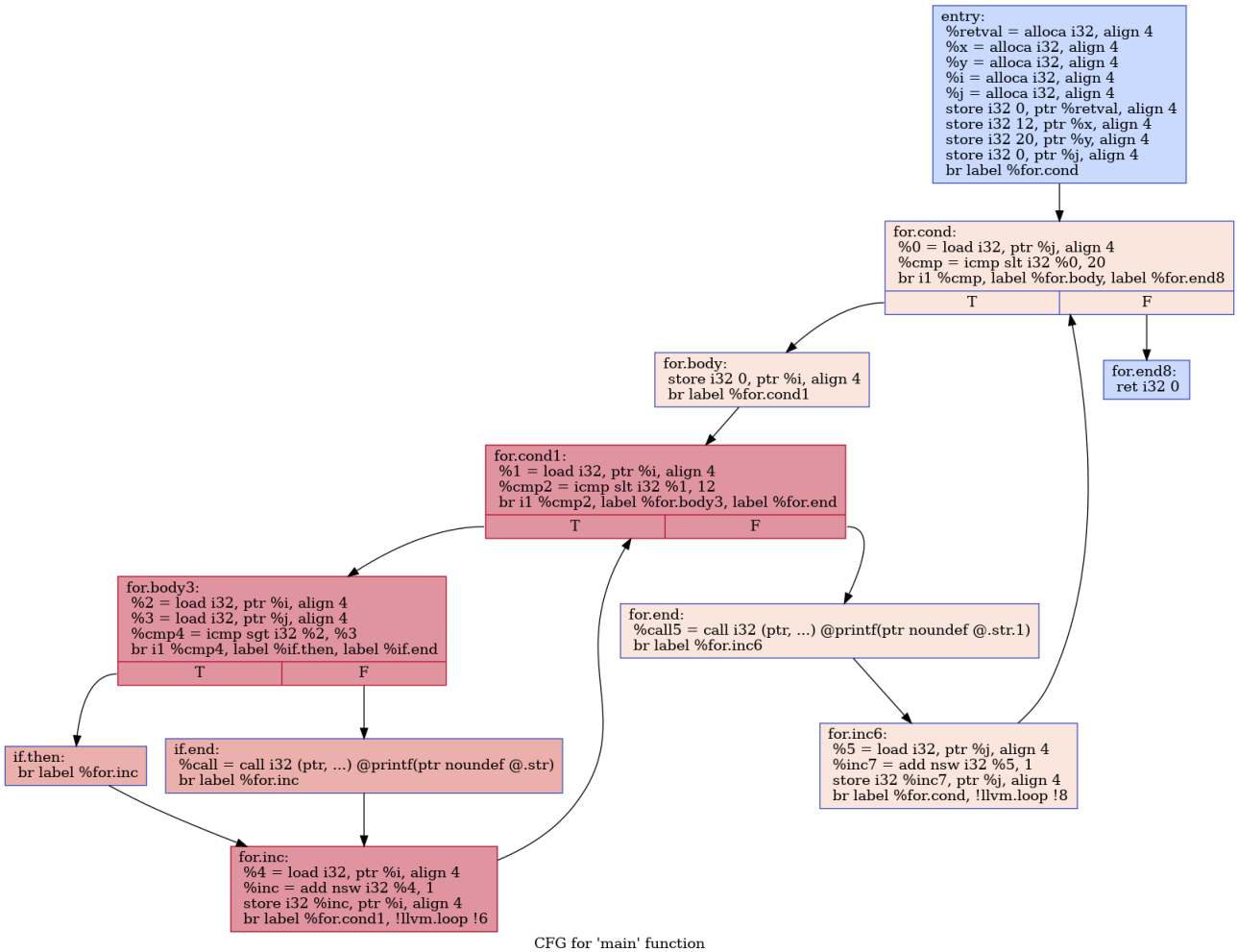

Figure 1: CFG example