Popcorn Compiler Internals 101:

Assignment #0– Setting up the Popcorn compiler toolchain

September 10, 2018

Contents

Introduction

The purpose of this assignment is to install the Popcorn compiler toolchain to build applications for Popcorn Linux. The Popcorn compiler toolchain is distributed as part of Popcorn Linux, and is responsible for building multi-ISA binaries suitable for migration between heterogeneous-ISA machines. The toolchain is built on top of several pieces of open-source software, most notably clang/LLVM (see **Licenses** for a complete list). The compiler supports POSIX-compliant standard C/C++ applications and includes an OpenMP [8] runtime suitable to enable multithreaded cross-node execution. For a more detailed description of the toolchain and what it supports, see [7, 1]

Disclaimer

The toolchain is a research prototype, and as such is unstable. The compiler is being updated constantly; be sure to check the master branch of the repository for bug-fixes and new features.

Use at your own peril and sanity!

Prerequisites

We assume the person installing the Popcorn compiler toolchain (hereafter creatively referred to as "user") is installing and running the toolchain on an x86-64 machine. Although in theory it should work on other machines (you'll need a working C and C++ compiler for all target architectures, preferably **`gcc`**/**`g++`**), the toolchain's installation and use has only been tested on x86-64. Note that although the toolchain is run on an x86-64 machine (the "host" in traditional autoconf parlance), it builds multi-ISA binaries to be run on multiple ISAs (the "targets"). In essence, it is a cross-compiler.

We assume the following "host", i.e., the environment on which the toolchain will be installed and used to build multi-ISA binaries:

- Ubuntu 16.04 LTS "Xenial Xerus" or later on x86-64
- Debian 8.8 "Jessie" or later on x86-64

We use these systems because their package managers have all the necessary tools for installation readily available. If you use a different GNU/Linux distribution you'll need to install equivalent versions or possibly even build them from source; your mileage may vary.

Learning Objectives

After completing the assignment, you should have accomplished the following:

1. Installed the Popcorn compiler toolchain (Problem 1)

2. Verified that clang/LLVM have Popcorn-specific functionality enabled (Problem 2)
3. Explored the installation directory, including where the toolchain's binaries/utilities/ISA-specific libraries reside (Problem 3)

Licenses

Onto the boring but necessary stuff. The Popcorn compiler toolchain is built on top of the following free and open-source software:

- clang/LLVM version 3.7.1 [6], released under the University of Illinois/NCSA Open Source License
- GNU gold version 2.27 [5], released under the GNU General Public License
- musl-libc version 1.1.18 [3], released under the MIT License
- libelf version 0.8.13 [9], released under the GNU Library General Public License
- libgomp packaged with gcc version 7.2.0 [4], released under the GNU General Public License

Problem #1: Installing the compiler toolchain

The Popcorn compiler repository is hosted on SSRG's organization GitHub page, located at `https://github.com/ssrg-vt/popcorn-compiler`. The repository is publicly-accessible with anonymous read-only access. **In order to get write access, you'll either need to be a part of the compiler team in the SSRG GitHub, or have an SSRG administrator add you as an outside collaborator**. Contact Rob Lyerly with your name and GitHub username to be added as an outside collaborator.

1. First, install prerequisite packages needed to build all components of the compiler toolchain.

```
$ sudo apt-get install build-essential gcc-aarch64-linux-gnu flex bison \
                       subversion cmake
```

   A few notes about why these packages are required:

   - The toolchain requires **gcc** for each ISA to be targeted by multi-ISA binaries, as **gcc** provides supporting libraries needed to build complete applications. For now, each new supported ISA requires installing **gcc-<ISA>-linux-gnu**, e.g., **gcc-powerpc64le-linux-gnu** for little-endian POWER8.
   - **build-essential** provides **gcc**, **g++**, **make** and several development libraries. **g++** is needed to compile clang/LLVM, and build systems for many of the components in the toolchain are driven using **make**.
   - **bison**/**flex** are used during the installation of binutils.
   - **subversion**/**cmake** are required for downloading and building clang/LLVM

2. After you've created your GitHub account and have been given access to the repo, clone it into a local directory:

```
$ git clone git@github.com:ssrg-vt/popcorn-compiler.git popcorn-compiler
Cloning into 'popcorn-compiler'...
...(enter your credentials)...
...(git processing)...

$ cd popcorn-compiler && ls
APPLICATIONS   common   install_compiler.py  lib      README  TODO  tutorial
BUG_REPORTING  INSTALL  ISSUES   lib         patches  test    tool  util
```

3. Finally, install the toolchain using the **`install_compiler.py`** script in the root of the repository. To start, list all available options with `-h`.

```
$ ./install_compiler.py -h
usage: install_compiler.py [-h] [--base-path BASE_PATH]
                           [--install-path INSTALL_PATH] [--threads THREADS]
                           [--skip-prereq-check] [--install-all]
                           [--install-llvm-clang] [--install-binutils]
                           [--install-musl] [--install-libelf]
                           [--install-libopenpop] [--install-stack-transform]
                           [--install-migration] [--install-stack-depth]
                           [--install-tools] [--install-utils]
                           [--install-namespace] [--targets TARGETS]
                           [--debug-stack-transformation]
                           [--libmigration-type {env_select,native,debug}]
                           [--enable-libmigration-timing]

...(explanation of options)...
```

There are a large number of options for configuration, many of which are used to select which components of the toolchain are installed. If in the future you need to reinstall a subset of components you can selectively enable installing certain parts of the toolchain using these options[1]. For now, we need to install all parts of the compiler so we specify where to install the toolchain and that the stack transformation library should be built to enable debugging output.

```
$ ./install_compiler.py --install-path <install folder> --install-all \
                        --debug-stack-transformation
...(lots of output)...
```

Depending on the beefiness of your machine (hopefully multi-core with lots of RAM), this could take a half hour or so. Go get a coffee in the meantime.

At this point the entire toolchain should be installed in the directory `<install folder>`. You can use the the default installation folder, `/usr/local/popcorn`, but on most systems you'll need super-user privileges to do so. Note that you'll need the debug build of the stack transformation library for later homeworks, but for a production environment where performance matters you'll want to build the library without debugging as it dumps a lot of information to disk.

At this point the compiler should be installed – if you had any errors, you'll need to investigate at which phase the install script died. As a convenience, you may want to add the folder containing the toolchain's binaries to your path for ease of reference (or, for example inside your `.bashrc`):

[1] Many of the libraries and tools are driven using simple Makefiles. The user can modify/reinstall a library or tool from directly inside its source folder in the Popcorn compiler source tree.

```
$ export PATH=$PATH:<install folder>/bin
```

Problem #2: Verifying Popcorn-specific functionality in clang/LLVM

Let's verify that clang/LLVM have been installed properly and that they have Popcorn-specific functionality enabled for building multi-ISA binaries.

Investigate clang's command-line options

Print the available command-line options from **clang** relevant to Popcorn and write them in the space below:

```
$ <install folder>/bin/clang -help | grep "popcorn"
```

What these flags do are a subject of later homeworks; for now, it's enough to verify that **clang** recognizes them.

View available targets

Print out all targets supported by the LLVM installation using **llc**, a tool that generates ISA-specific machine code from ISA-agnostic LLVM intermediate representation ("LLVM IR") and write them in the space below:

```
$ <install folder>/bin/llc -version
LLVM (http://llvm.org/):
  LLVM version 3.7.1
  DEBUG build with assertions.
  Built Nov 27 2017 (20:51:52).
  Default target: x86_64-unknown-linux-gnu
  Host CPU: haswell

  Registered Targets:
    ...(available targets)...
```

You can also use **llc** to get detailed target-specific information, including options for tuning the backend to optimize for specific CPUs and features:

```
# Format: llc -march <ISA> -mattr=help, e.g.,
$ <install folder>/bin/llc -march aarch64 -mattr=help
Available CPUs for this target:

cortex-a53 - Select the cortex-a53 processor.
cortex-a57 - Select the cortex-a57 processor.
cortex-a72 - Select the cortex-a72 processor.
cyclone    - Select the cyclone processor.
generic    - Select the generic processor.

Available features for this target:

a53      - Cortex-A53 ARM processors.
a57      - Cortex-A57 ARM processors.
crc      - Enable ARMv8 CRC-32 checksum instructions.
crypto   - Enable cryptographic instructions.
cyclone  - Cyclone.
fp-armv8 - Enable ARMv8 FP.
neon     - Enable Advanced SIMD instructions.
v8.1a    - Support ARM v8.1a instructions.
zcm      - Has zero-cycle register moves.
zcz      - Has zero-cycle zeroing instructions.

Use +feature to enable a feature, or -feature to disable it.
For example, llc -mcpu=mycpu -mattr=+feature1,-feature2
'+help' is not a recognized feature for this target (ignoring feature)
```

Verify Popcorn-specific middle-end passes

LLVM drives the compilation process by applying a set of passes to analyze and transform code in the middle-end (ISA-agnostic) and back-end (ISA-specific). There are several Popcorn-specific passes in both phases to prepare applications for migration. The passes in the middle-end are exposed via **opt**, LLVM's ISA-agnostic optimizer (the passes in the back-end have no such tool). Print all available passes in **opt**:

```
$ <install folder>/bin/opt -help
...(lots of output)...
```

You should see the following passes:

- `-select-migration-points` – the compiler is responsible for inserting call-outs to a migration library in order to invoke cross-node migration (see [7, 1] for more details). This pass selects locations at which to insert migration points.
- `-migration-points` – inserts call-outs to the migration library at points selected by the previous pass
- `-live-values` – runs a live-value analysis over the IR, which determines the set of all function-local variables that are live at a give location inside of the function. See [2] for more information.
- `-insert-stackmaps` – the compiler uses LLVM stackmaps to trigger stack transformation metadata generation, the subject of future homeworks. This pass uses the liveness analysis to instruct the back-end to capture where live values are located in each ISA-specific stack frame.
- `-libc-stackmaps` – similar to `-insert-stackmaps`, but hard-codes a few specific cases in musl-libc required for terminating stack transformation.

Problem #3: Exploring the installation directory

Let's take a deeper look into the installation of the Popcorn compiler:

```
$ ls <install folder>
aarch64  bin  include  lib  share  src  x86_64  x86_64-pc-linux-gnu
```

- `src` contains the patched source code for clang/LLVM and binutils; the Popcorn compiler applies a set of patches to downloaded source rather than forking the entire repositories. You can modify, rebuild and reinstall the components from inside the build folder of each subdirectory, e.g., `<install folder>/src/llvm/build`. The source for all other Popcorn components lives in the repository you cloned earlier.
- `bin` contains the executables for clang/LLVM, binutils, and Popcorn-specific alignment/post-processing utilities. This is where the components comprising the Popcorn compiler toolchain reside in your system. These may also reference headers and libraries in `include` and `lib`.
- Finally, there is a sub-folder for each target ISA supported by the toolchain, e.g., `aarch64` and `x86_64`. Each of these folders contains the cross-compilation environment required for each target in the multi-ISA binary (how these are used is the subject of future homeworks). Inside each target subdirectory, you'll find headers, libraries and musl-specific build tools. musl provides most of this, as it provides the C library used by virtually all applications, `libc.a`. Additionally, you'll find `libelf.a`, `libmigrate.a` and `libstack-transform.a` compiled for each architecture. How the Popcorn compiler uses these target subdirectories is beyond the scope of this assignment, but it's important to note that when linking against new libraries, the user must provide a version compiled for each target and placed inside the relevant target folder inside the installation.

References

[1] Barbalace, A., Lyerly, R., Jelesnianski, C., Carno, A., Chuang, H.-R., Legout, V., and Ravindran, B. Breaking the boundaries in heterogeneous-isa datacenters. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2017), ASPLOS '17, ACM, pp. 645–659.

[2] Brandner, F., Boissinot, B., Darte, A., De Dinechin, B. D., and Rastello, F. *Computing Liveness Sets for SSA-Form Programs.* PhD thesis, INRIA, 2011.

[3] Felker, R. musl libc. `http://www.musl-libc.org/`, 2017.

[4] GNU. GNU libgomp. `https://gcc.gnu.org/onlinedocs/libgomp/`, 2018.

[5] GNU/Free Software Foundation. Binutils - GNU Project - Free Software Foundation. `https://www.gnu.org/software/binutils/`, 2017.

[6] Lattner, C., and Adve, V. LLVM: a compilation framework for lifelong program analysis transformation. In *International Symposium on Code Generation and Optimization, 2004. CGO 2004.* (March 2004), pp. 75–86. Project site – `http://llvm.org/`.

[7] Lyerly, R. F. *Popcorn Linux: A Compiler and Runtime for State Transformation Between Heterogeneous-ISA Architectures.* PhD thesis, Virginia Polytechnic Institute and State University, 2016.

[8] OpenMP Architecture Review Board. OpenMP application programming interface. `http://www.openmp.org/wp-content/uploads/openmp-4.5.pdf`, November 2015.

[9] Riepe, M. Libelf - free software directory. `https://directory.fsf.org/wiki/Libelf`, 2017.