Fighting Non-determinism in C++ Compilers

Mandeep Singh Grang
Qualcomm Innovation Center, Inc.
mgrang@quicinc.com
https://github.com/mgrang

INTRODUCTION

A C++ compiler exhibits non-deterministic behavior if, for the same input program, the object code generated by the compiler differs from run to run. In this poster, we first explore the causes of such non-determinism. Then we outline the scenarios where non-determinism is observed and examine why such behavior is undesirable. We then present a case study on our work to uncover and fix non-deterministic behavior in the LLVM compiler. Finally, we report on the impact that our work has had on the LLVM community, the total number of bugs found and how we fixed them.

RELEVANCE

Millions of C++ developers around the world use compilers to develop their programs. Not every developer necessarily understands the internals of the compiler. So, having a robust compiler becomes supremely important. However, the behavior of a compiler may not always be deterministic. For the same input program, it may generate different code in different scenarios. This non-determinism can make debugging difficult, result in hard-to-reproduce bugs, cause unexpected runtime crashes or unpredictable performance.

Our work attempts to uncover non-deterministic behavior in the LLVM C++ compiler thereby making LLVM more robust.

DISCUSSION

A C++ compiler may exhibit non-deterministic behavior. This means that for the same input program the object code generated by the compiler may differ from run to run.

This non-deterministic behavior can either remain hidden or manifest itself in several scenarios. For example, the same compiler hosted on different operating systems might generate different object code for the same input program. Or there might be differences in behavior between asserts and non-asserts version of the same compiler. Or even back-to-back runs of the same compiler can produce different object code for the same input.

We have identified three main causes of non-deterministic behavior in a C++ compiler: iteration of unordered containers, hashing of pointer keys, and the use of non-stable sort functions. All three arise due to poor understanding of the behavior of various containers and algorithms.

The detection of such non-deterministic behavior is often challenging since the compiler may not always behave in an expected way. In LLVM we try to uncover non-determinism in 2 ways:

1. Iteration order non-determinism
We implemented a "reverse iteration" mode for all supported unordered containers in LLVM. The CMake flag LLVM_REVERSE_ITERATION enables the reverse iteration mode. This mode makes all supported containers iterate in reverse, by default. The idea is to compare the output of a reverse iteration compiler with that of a forward iteration compiler to weed out iteration order randomness. This mode is transparent to the user and comes with almost zero runtime cost. The following upstream buildbot tracks this mode: http://lab.llvm.org:8011/builders/reverse-iteration

2. Sorting order non-determinism
We added a wrapper function to LLVM called llvm::sort which randomly shuffles a container before invoking std::sort. The idea is that randomly shuffling a container would weed out non-deterministic sorting order of keys with the same values. The following upstream buildbot tracks this mode: http://lab.llvm.org:8011/builders/llvm-clang-x86_64-expensive-checks-win

We also outline some best practices we followed in LLVM to avoid or fix non-deterministic behavior:

1. Sort the container before iteration
2. Use a stronger sort predicate
3. Use a stable sort function
4. Use an ordered container

COMPLETION STATUS

Our work to enable reverse iteration and random shuffling a container is complete and available upstream in the latest 6.0 release of LLVM. We have so far uncovered and fixed 42 iteration order bugs and 44 sorting order bugs. The upstream buildbots regularly catch non-determinism bugs and the community promptly fixes them. As a result of our work, the LLVM community has become more diligent in their use of containers and sorting algorithms. We have also added coding standards for LLVM compiler developers on the correct use of unordered containers and sorting algorithms:
[1] https://llvm.org/docs/CodingStandards.html#beware-of-non-determinism-due-to-ordering-of-pointers
[2] https://llvm.org/docs/CodingStandards.html#beware-of-non-deterministic-sorting-order-of-equal-elements

REFERENCES

Our work has featured several times in the LLVM weekly newsletters and other places:
[1] https://bugs.swift.org/browse/SR-6154
[2] https://blog.jetbrains.com/clion/2017/12/cpp-annotated-sep-dec-2017
[3] http://bitupdate.us/compiler-infrastructure-llvm-5-0-provides-new-tools
[4] http://llvmweekly.org/issue/224
[5] http://llvmweekly.org/issue/201
[6] http://llvmweekly.org/issue/193
[7] http://llvmweekly.org/issue/192
[8] http://llvmweekly.org/issue/184
[9] http://llvmweekly.org/issue/151
[10] http://lists.llvm.org/pipermail/llvm-dev/2016-November/107098.html
[11] http://lists.llvm.org/pipermail/llvm-dev/2017-July/115025.html
[12] http://lists.llvm.org/pipermail/llvm-dev/2017-August/116975.html
[13] http://lists.llvm.org/pipermail/llvm-dev/2017-October/118639.html

BIO

I am a Compiler Engineer with a Master's in Computer Science from Stony Brook University, NY. I have been actively contributing to the open source LLVM project for the past 7 years. I have expertise in C++, Python, and Bash. I develop mainly for the ARM, AArch64, and RISCV targets. I currently work as a Senior Engineer at Qualcomm Innovation Center, Inc. in San Diego, CA.