Path based test coverage for quick test feedback

February 04, 2023

Agenda

- Introduction
- LLVM testing suite
- LLVM source-based code coverage
- Implementation
- Demo

Introduction

Most of the day to day tests in LLVM are regression tests executed by Lit, structured as source code or IR to be passed to some binary, rather than test code directly calling the code to be tested.

The goal of this work was to help developers create good test coverage for their patch and enable reviewers to verify that they have done so.

To accomplish this we have introduced a tool that can be fed a patch as input, add coverage instrumentation for the affected source files, runs Lit tests, and records which test cases cause each counter of source code to be executed.

LLVM Testing Suite
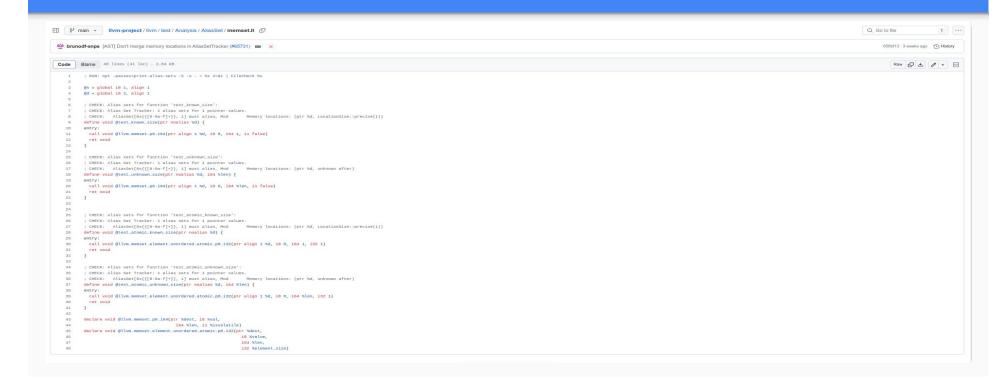
Regression tests

The regression tests are small pieces of code that test a specific feature of LLVM or trigger a specific bug in LLVM. The language they are written in depends on the part of LLVM being tested. These tests are driven by the Lit testing tool (which is part of LLVM), and are located in the llvm/test directory.
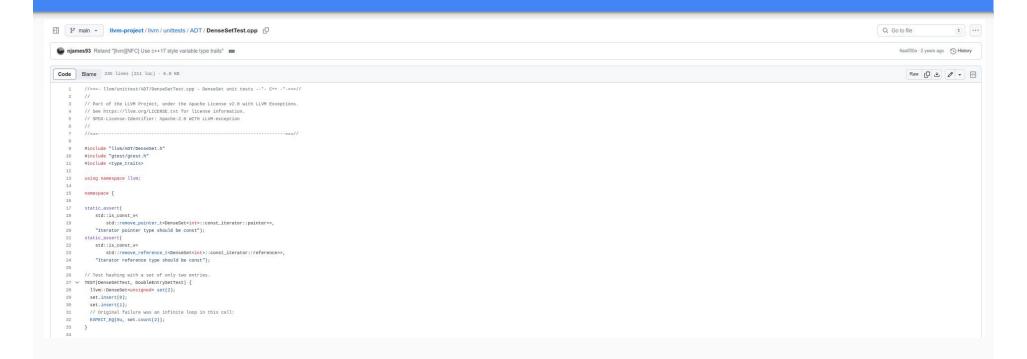
Unit tests

Unit tests are written using Google Test and Google Mock and are located in the llvm/unittests directory. In general unit tests are reserved for targeting the support library and other generic data structure, we prefer relying on regression tests for testing transformations and analysis on the IR.

Regression test

Unit test

njames93 Reland "[llvm][NFC] Use c++17 style variable type traits"   6aa050a · 2 years ago History

Code   Blame   235 lines (211 loc) · 6.8 KB   Raw

```cpp
 1   //===- llvm/unittest/ADT/DenseSetTest.cpp - DenseSet unit tests --*- C++ -*-===//
 2   //
 3   // Part of the LLVM Project, under the Apache License v2.0 with LLVM Exceptions.
 4   // See https://llvm.org/LICENSE.txt for license information.
 5   // SPDX-License-Identifier: Apache-2.0 WITH LLVM-exception
 6   //
 7   //===----------------------------------------------------------------------===//
 8
 9   #include "llvm/ADT/DenseSet.h"
10   #include "gtest/gtest.h"
11   #include <type_traits>
12
13   using namespace llvm;
14
15   namespace {
16
17   static_assert(
18       std::is_const_v<
19           std::remove_pointer_t<DenseSet<int>::const_iterator::pointer>>,
20       "Iterator pointer type should be const");
21   static_assert(
22       std::is_const_v<
23           std::remove_reference_t<DenseSet<int>::const_iterator::reference>>,
24       "Iterator reference type should be const");
25
26   // Test hashing with a set of only two entries.
27   TEST(DenseSetTest, DoubleEntrySetTest) {
28       llvm::DenseSet<unsigned> set(2);
29       set.insert(0);
30       set.insert(1);
31       // Original failure was an infinite loop in this call:
32       EXPECT_EQ(0u, set.count(2));
33   }
34
```

LLVM source-based code coverage

The code coverage workflow consists of three main steps:

Compiling with coverage enabled.
clang++ -fprofile-instr-generate -fcoverage-mapping foo.cc -o foo

Running the instrumented program.
LLVM_PROFILE_FILE="foo.profraw" ./foo

Creating coverage reports.
llvm-profdata merge -sparse foo.profraw -o foo.profdata
llvm-cov show ./foo -instr-profile=foo.profdata

Sample coverage report

```
xgupta@linux ~ $ /usr/bin/llvm-cov show ./test -instr-profile=test.profdata

    1|       |#include <stdio.h>
    2|       |
    3|      1|int main() {
    4|      1|    int num;
    5|       |
    6|       |    // Input a number
    7|      1|    printf("Enter an integer: ");
    8|      1|    scanf("%d", &num);
    9|       |
   10|       |    // Check if the number is even or odd using if-else loop
   11|      1|    if (num % 2 == 0) {
   12|      0|        printf("%d is an even number.\n", num);
   13|      1|    } else {
   14|      1|        printf("%d is an odd number.\n", num);
   15|      1|    }
   16|       |
   17|      1|    return 0;
   18|      1|}
   19|       |
```

Implementation

https://reviews.llvm.org/D154280 ([LIT] Added an option to llvm-lit to emit the necessary test coverage data, divided per test case)

https://github.com/llvm/llvm-project/pull/71841 ([Code Coverage] Add a tool to check test coverage of a patch)

Functions implemented for tool

1. configure_logging
2. custom_print
3. create_patch_from_last_commit
4. extract_source_files_from_patch
5. write_source_file_allowlist
6. extract_modified_source_lines_from_patch
7. build_llvm
8. run_single_test_with_coverage
9. run_modified_lit_tests
10. run_modified_unit_tests
11. process_coverage_data
12. read_coverage_file_and_return_result
13. print_coverage_details
14. print_common_uncovered_lines
15. report_covered_and_uncovered_lines
16. parse_suite_info
17. find_lit_tests
18. parse_args
19. main

GitHub CI workflow


```yaml
.github/workflows/patch-based-code-coverage.yml

@@ -0,0 +1,43 @@
 1  + name: LLVM Code Coverage
 2  +
 3  + on:
 4  +   push:
 5  +     branches:
 6  +       - test-clang
 7  +
 8  + jobs:
 9  +   code-coverage:
10  +     name: Generate Code Coverage Report
11  +     runs-on: self-hosted
12  +
13  +     steps:
14  +       - name: Checkout LLVM Repository
15  +         uses: actions/checkout@v1
16  +
17  +       - name: Install LLVM project build dependency
18  +         run: |
19  +           sudo apt-get install llvm
20  +           sudo apt-get install clang
21  +           sudo apt-get install lld
22  +           sudo apt-get install cmake
23  +           sudo apt-get install ninja-build
24  +           sudo apt-get install -y python3.11
25  +           pip install argparse unidiff
26  +
27  +       - name: Build LLVM project
28  +         run: |
29  +           mkdir build
30  +           cd build
31  +           cmake -G Ninja ../llvm -DLLVM_ENABLE_PROJECTS="clang" -DCMAKE_BUILD_TYPE=Release -DLLVM_USE_LINKER=lld -DCMAKE_C_COMPILER=/usr/local/bin/clang -DCMAKE_CXX_COMPILER=/usr/local/bin/clang++ -DLLVM_TARGETS_TO_BUILD="X86;AArch64;RISCV"  -DLLVM_ENABLE_ASSERTIONS=ON
32  +           ninja
33  +
34  +       - name: Run the patch-based code coverage tool
35  +         run: |
36  +           cp llvm/utils/git-check-coverage .
37  +           python3 git-check-coverage -b build bin/clang clang/test
38  +
39  +       - name: Archive code coverage artifacts
40  +         uses: actions/upload-artifact@v2
41  +         with:
42  +           name: code-coverage-results
43  +           path: build/patch_coverage.log
```

Demo

https://github.com/llvm/llvm-project/pull/71894 ([Clang] add user-level sizeless attribute)

https://github.com/xgupta/llvm-project/actions/runs/6994512086/job/19028363714#step:5:3726

https://github.com/llvm/llvm-project/pull/72273 ([AArch64] Add an AArch64 pass for loop idiom transformations)

https://github.com/xgupta/llvm-project/actions/runs/6994498992/job/19028333944#step:5:2609

Result

```
2607
2608  Source files in the patch (excluding test files):
2609  llvm/lib/Analysis/TargetTransformInfo.cpp
2610  llvm/lib/Target/AArch64/AArch64LoopIdiomTransform.cpp
2611  llvm/lib/Target/AArch64/AArch64TargetMachine.cpp
2612
2613
2614
2615  Common uncovered lines for llvm/lib/Analysis/TargetTransformInfo.cpp after all changes:
2616
2617  Common uncovered lines for llvm/lib/Target/AArch64/AArch64LoopIdiomTransform.cpp after all changes:
2618    Line 104: return false;
2619    Line 131: Pass *llvm::createAArch64LoopIdiomTransformPass() {
2620    Line 132: return new AArch64LoopIdiomTransformLegacyPass();
2621    Line 133: }
2622    Line 138: LPMUpdater &) {
2623    Line 139: if (DisableAll)
2624    Line 140: return PreservedAnalyses::all();
2625    Line 142: const auto *DL = &L.getHeader()->getModule()->getDataLayout();
2626    Line 144: AArch64LoopIdiomTransform LIT(&AR.DT, &AR.LI, &AR.TTI, DL);
2627    Line 145: if (!LIT.run(&L))
2628    Line 146: return PreservedAnalyses::all();
2629    Line 148: return PreservedAnalyses::none();
2630    Line 149: }
2631    Line 161: return false;
2632    Line 166: return false;
2633    Line 197: return false;
2634    Line 207: return false;
2635    Line 211: return false;
2636    Line 224: return false;
2637    Line 239: return false;
2638    Line 298: return false;
2639    Line 304: return false;
2640    Line 319: return false;
2641    Line 406: CurLoop->getParentLoop()->addChildLoop(SVELoop);
2642    Line 407: CurLoop->getParentLoop()->addChildLoop(ScalarLoop);
2643    Line 696: ResPhi = true;
2644    Line 701: PN.addIncoming(ByteCmpRes, CmpBB);
2645    Line 722: CurLoop->getParentLoop()->addBasicBlockToLoop(CmpBB, *LI);
2646
2647  Common uncovered lines for llvm/lib/Target/AArch64/AArch64TargetMachine.cpp after all changes:
2648    Line 543: void AArch64TargetMachine::registerPassBuilderCallbacks(PassBuilder &PB) {
2649    Line 544: PB.registerLateLoopOptimizationsEPCallback(
2650    Line 545: [=](LoopPassManager &LPM, OptimizationLevel Level) {
2651    Line 546: LPM.addPass(AArch64LoopIdiomTransformPass());
2652    Line 547: });
2653    Line 548: }
```

Questions ?

Write to me on shivam98.tkg@gmail.com

or connect