

# 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.



# Unit test

```
main - llvm-project / llvm / unittests / ADT / DenseSetTest.cpp
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
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.profrw" ./foo
```

Creating coverage reports.

```
llvm-profdata merge -sparse foo.profrw -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

```
43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000
```

# 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 148: 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(SVLoop);
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.registerLateLoopOptimizationEPCCallback(
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](mailto:shivam98.tkg@gmail.com)

or connect

