AIM OF THIS TUTORIAL

- Newcomers looking to start working on LLVM
- Developers who want additional information about the testing infrastructure
- Anyone would is looking to contribute to improving the test infrastructure
WHAT YOU WILL LEARN

- Write comprehensive tests for your contributions to LLVM
- Run tests to catch bugs locally before committing
- Understand how to collect compile and performance timings to understand the impact of your proposed changes. Supply data to support your pull request
OUTLINE

Tests

- Unit Tests
- Regression Tests
- Debug Info Tests
- Whole Program Tests
OUTLINE

Tools and Frameworks

- Unit Tests
  - Google Test
- Regression Tests
  - FileCheck
  - Lit
- Debug Info Tests
- Whole Program Tests
  - Google Benchmark
- LNT
- Build Bots
UNIT TESTS

GOOGLE TEST
UNIT TESTS

- Level of software testing aimed to validate that individual units/components perform as designed
- llvm-project/llvm/unittests

```
make check-llvm-unit
llvm-lit: llvm/utils/lit/lit/main.py:502: note ...
Testing Time: 9.82s
Expected Passes : 3772
[100%] Built target check-llvm-unit
```
UNIT TESTS

LLVM Example

// Check that a function arg can't trivially alias a global when we're accessing
// >sizeof(global) bytes through that arg, unless the access size is just an
// upper-bound.

TEST_F(BasicAAATest, AliasInstWithObjectOfImpreciseSize) {
    {...}
    ASSERT_EQ(BasicAA.alias(MemoryLocation(IncomingI32Ptr, LocationSize::precise(4)),
                              MemoryLocation(GlobalPtr, LocationSize::precise(1)), AAQI),
               AliasResult::NoAlias);
}
UNIT TESTS
Google Benchmark Macros

Basic Assertions

These assertions do basic true/false condition testing.

<table>
<thead>
<tr>
<th>Fatal assertion</th>
<th>Nonfatal assertion</th>
<th>Verifies</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASSERT_TRUE(condition);</td>
<td>EXPECT_TRUE(condition);</td>
<td>condition is true</td>
</tr>
<tr>
<td>ASSERT_FALSE(condition);</td>
<td>EXPECT_FALSE(condition);</td>
<td>condition is false</td>
</tr>
</tbody>
</table>

- There is also support for binary and string comparison assertions
UNIT TESTS

Google Test concepts

- Test
- Test Suite (group of related tests)

- Test Fixtures (Same data multiple tests)
  - Setup()
  - TearDown()
UNIT TESTS

LLVM Example

// Check that a function arg can't trivially alias a global when we're accessing
// >sizeof(global) bytes through that arg, unless the access size is just an
// upper-bound.

TEST_F(BasicAATest, AliasInstWithObjectOfImpreciseSize) {

    {...}

    ASSERT_EQ(
        BasicAA.alias(MemoryLocation(IncomingI32Ptr, LocationSize::precise(4)),
                       MemoryLocation(GlobalPtr, LocationSize::precise(1)), AAQI),
        AliasResult::NoAlias);

}
UNIT TESTS

LLVM Example

// Check that a function arg can't trivially alias a global when we're accessing
// sizeof(global) bytes through that arg, unless the access size is just an
// upper-bound.

TEST_F(BasicAA, AliasInstWithObjectOfImpreciseSize) {
    {...}
    ASSERT_EQ(
        BasicAA.alias(MemoryLocation(IncomingI32Ptr, LocationSize::precise(4)),
                      MemoryLocation(GlobalPtr, LocationSize::precise(1)), AAQI),
        AliasResult::NoAlias);
}
REGRESSION TESTS
REGRESSION TESTS

- Small pieces of code that test a specific feature or trigger a specific bug in LLVM.
- Written in various languages depending on what is being tested. (C/C++, LLVM IR, etc)

    llvm-project/llvm/test

- Great Documentation

    ; RUN: opt < %s -basicaa -aa-eval -print-all-modref-info -disable-output 2>&1 | FileCheck %s
REGRESSION TESTS

How to Run

make check-llvm

llvm-lit -v llvm-project/llvm/test/Analysis/BasicAA/noalias-geps.ll
-- Testing: 1 tests, single process –
PASS: LLVM :: Analysis/BasicAA/noalias-geps.ll (1 of 1)
Testing Time: 0.99s
Expected Passes : 1
LIT – LLVM INTEGRATED TESTER
LIT

- lit is a tool for executing LLVM and Clang style test suites
- Provides a summary of results and information on failures
- Configurable

Test Discovery
  - lit recursively searches for tests based on the configuration
  - lit can also recursively find full test suites
LIT

Options

\texttt{llvm-lit [options] path/to/test/or/directory/with/tests}

- Many options to control execution
  - Set the number of testing threads “-j N, --threads N”
  - Filter tests based on regular expression “--filter REGEX”

- \texttt{lit} has support for running tests under \texttt{valgrind}
  - “--vg, --vg-leak, --vg-arg <ARG>”

\texttt{llvm-project/llvm/utils/lit}
PASS: A (1 of 4)
PASS: B (2 of 4)
FAIL:   C (3 of 4)

*************** TEST 'C' FAILED ***************
Test 'C' failed as a result of exit code 1.

***************
PASS: D (4 of 4)
FILECHECK
FILECHECK

- Flexible pattern matching file verifier

- Takes in two files and uses one to verify the other

- Useful to verify the output of a tool
  - `clang --cc1 --emit-llvm <...> | filecheck verification_file`

- Optimized for matching multiple different inputs in one file with a specific order
When fixed string matching is not sufficient, FileCheck supports using regular expressions

; CHECK: movhpd {{[0-9]+}}(%esp), {{%xmm[0-7]}}

It is useful to verify that a matched pattern occurs again later in the file

; CHECK: op [[REG:r[0-9]+]], [[REG]]
FILECHECK

CHECK

- Check for fixed strings that must occur in order
- Ignores horizontal whitespace differences

```c
define void @sub1(i32* %p, i32 %v) {
  entry:
  ; CHECK: sub1:
  ; CHECK: subl
  %0 = tail call i32 @llvm.atomic.load.sub.i32.p0i32(i32* %p, i32 %v)
  ret void
}
```
FILECHECK
CHECK-NEXT

- Checks that matches occur on exactly consecutive lines

; CHECK:       t2:
; CHECK:       movl 8(%esp), %eax
; CHECK-NEXT:  movapd (%eax), %xmm0
; CHECK-NEXT:  movhpd 12(%esp), %xmm0
; CHECK-NEXT:  movl 4(%esp), %eax
FILECHECK

CHECK-NOT

- Verifies that a string does NOT occur between two matches.

- Very useful in combination with other Checks

; CHECK: @coerce_offset0
; CHECK-NOT: load
; CHECK: ret i8
FILECHECK

CHECK-SAME

- Allows you to verify that matches happen on the same line as the previous match

!0 = !DILocation(line: 5, scope: !1, inlinedAt: !2)

; CHECK: !DILocation(line: 5,
; CHECK-NOT: column:
; CHECK-SAME: scope: ![[SCOPE:[0-9]+]]
FILECHECK
CHECK-SAME

- Allows you to verify that matches happen on the same line as the previous match
- Useful with CHECK-NOT

!0 = !DILocation(line: 5, scope: !1, inlinedAt: !2)

; CHECK: !DILocation(line: 5,
; CHECK-NOT: column:
; CHECK-SAME: scope: ![[SCOPE:[0-9]+]]
FILECHECK

CHECK-EMPTY

- Checks that next line has nothing on it, not even whitespace

declare void @foo()

declare void @bar()

; CHECK: foo
; CHECK-EMPTY:
; CHECK-NEXT: bar
FILECHECK

CHECK-COUNT-<NUM>

- Checks that same pattern occurs over and over again

Loop at depth 1
Loop at depth 1
Loop at depth 1
Loop at depth 1
Loop at depth 2
Loop at depth 3

; CHECK-COUNT-6: Loop at depth {[[0-9]+]}
FILECHECK

CHECK-DAG

- Verify that matches occur in order, but allow for lines in between
- Need to be careful when defining and using variables

```cpp
struct Foo { virtual void method(); }
Foo f; // emit vtable
// CHECK-DAG: @_ZTV3Foo = ...
```

```cpp
struct Bar { virtual void method(); }
Bar b;
// CHECK-DAG: @_ZTV3Bar = ...
```
FILECHECK

CHECK-DAG

- Verify that matches occur in order, but allow for lines in between
- Need to be careful when defining and using variables
- Useful with CHECK-NOT

; CHECK-DAG: BEFORE
; CHECK-NOT: NOT
; CHECK-DAG: AFTER
FILECHECK

CHECK-LABEL

- Same as CHECK, but FileCheck assumes the directive cannot be matched elsewhere
- Useful for producing better error messages by dividing input into separate blocks
- Helps avoid issues with CHECK that match earlier than expected
FILECHECK

check-prefix

- Allows multiple test configurations to live in one .ll file.

; RUN: | FileCheck %s -check-prefix=X64

; X32: pinsrd_1:
; X32: pinsrd $1, 4(%esp), %xmm0

; X64: pinsrd_1:
; X64: pinsrd $1, %edi, %xmm0
DEBUG INFO TESTS
DEBUG INFO TESTS

- Collection of test to verify the debugging information generated by the compiler
  
  - Place into: clang/test
  - make test

- Includes debugger commands using the “DEBUGGER : ” prefix along with the intended output using the ”CHECK : ” prefix
define i32 @f1(i32 %i) nounwind ssp {
    ; DEBUGGER: break f1
    ; DEBUGGER: r
    ; DEBUGGER: p i
    ; CHECK: $1 = 42
    entry:
}
LLVM TEST SUITE

GOOGLE BENCHMARK
TEST SUITE

Structure

- Collection of whole program tests

- Lives separate from LLVM
  - [https://github.com/llvm-mirror/test-suite](https://github.com/llvm-mirror/test-suite)

- While every program can work as a correctness test, some are not suitable for measuring performance.
  - Use the “TEST_SUITE_BENCHMARKING_ONLY=ON” cmake option
TEST SUITE

Example Multi-source

- Test programs that are built with a single or multiple source files
- Includes large benchmarks and whole applications
- Tests are defined in CMakeLists.txt

```cpp
set(FP_TOLERANCE 0.00001)
list(APPEND CPPFLAGS -ffast-math \ 
    -DVERIFICATION_OUTPUT_ONLY=ON)
set(RUN_OPTIONS 450)
llvm_multisource(HACCKernels)
```

test-suite/
  - SingleSource/
  - MultiSource/
  - MicroBenchmarks/
  - External/
  - Bitcode/
  - CTMark/
TEST SUITE

Example Microbenchmark

- Lit allows reporting multiple results from one run
- Single executable that reports timing for multiple microbenchmarks

********** TEST 'test-suite :: Dilate.test (2 of 15) **********

**********

*** MICRO-TEST: BENCHMARK_DILATE/1024
exec_time: 9140.8000

*** MICRO-TEST: BENCHMARK_DILATE/128
exec_time: 137.2530

test-suite/
- SingleSource/
- MultiSource/
- MicroBenchmarks/
- External/
- Bitcode/
- CTMark/
TEST SUITE

Google Benchmark

- Library to generate quick performance benchmark tests
- Allows for the generation of multiple test sizes on a single code snippet
- Dynamically determines the number of iterations for the benchmark to ensure the ultimate result will be statistically stable
TEST SUITE

Google Benchmark

static void BM_VOL3D_CALC_RAW(benchmark::State& state) {
    {...}
    for( auto _ : state ) {
        for (Index_type i = domain.fpz ; i <= domain.lpz ; i++ ) {
            {...}
        }
    }
}

BENCHMARK(BM_VOL3D_CALC_RAW)->Arg(SHORT)->Arg(MEDIUM)->Arg(LONG)->Unit(benchmark::kMicrosecond);
TEST SUITE

Google Benchmark

```c++
static void BM_VOL3D_CALC_RAW(benchmark::State& state) {
    {…}
    for( auto _ : state) {
        for (Index_type i = domain.fpz ; i <= domain.lpz ; i++ ) {
            {…}
        }
    }

    BENCHMARK(BM_VOL3D_CALC_RAW)->Arg(SHORT)->Arg(MEDIUM)->
             Arg(LONG)->Unit(benchmark::kMicrosecond);
```
TEST SUITE

External Suites

- Contains support for running tests which cannot be directly distributed with the test-suite. Eg. SPEC

- Enabled by either:
  - Placing in “test-suite/test-suite-externals/xxx”
  - Using configuration option “-DTEST_SUITE_xxx_ROOT=“
TEST SUITE
Bitcode & CTMark

- Bitcode
  - Tests that are written in LLVM bitcode

- CTMark
  - Set of compile time benchmarks to measure compile time
  - Links to other benchmarks in other locations
  - Build with:
    -DTEST_SUITE_SUBDIRS=CTMARK
TEST SUITE
Profile Guided Optimization

# Profile generation run:

% cmake  -DTEST_SUITE_PROFILE_GENERATE=ON \ 
   -DTEST_SUITE_RUN_TYPE=train \ 
   ../test-suite

% make;  % llvm-lit .

# Use the profile data for compilation and actual benchmark run:

% cmake  -DTEST_SUITE_PROFILE_GENERATE=OFF \ 
   -DTEST_SUITE_PROFILE_USE=ON \ 
   -DTEST_SUITE_RUN_TYPE=ref \ 
   .
TEST SUITE
Adding a New Test

test-suite/MultiSource/CMakeLists.txt

  add_sudirectory(MyTest)  # Include when building the test suite

CMakeLists.txt
mytest.reference_output
sourcefile1.c
{...}
sourcefileN.c

# Set compile and run flags
# Output file to verify executable output
# All source files needed
LNT
LNT

- It is an infrastructure for performance testing
- Web application for accessing and visualizing performance data
- Command line utilities to generate and collect test results
- Utilizes an extensible format for exchanging data between the test producer and the server
LNT

```
sudo easy_install virtualenv
virtualenv ~/mysandbox
svn co http://llvm.org/svn/llvm-project/Lnt/trunk ~/Lnt
~/mysandbox/bin/python ~/Lnt/setup.py develop

Lnt runtest nt \
  --sandbox SANDBOX \
  --cc clang \
  --test-suite ~/path/to/llvm-test-suite
```
There are several ways to reduce the noise in the test results
- Run the benchmarks serially --threads 1
  - Can also compile serially for compile timing --build-threads 1
- Use perf to have more accurate timings --use-perf=1
- Pin the benchmark to a specific core --make-param="RUNUNDER=taskset --c 1"
- Collect multiple timing samples --multisample=10
You can collect your results and use the web application locally

# Create a local LNT instance
lnt create ~/myperfdb

# Import your test results either after or as part of the run
lnt import ~/myperfdb SANDBOX/test-<time-stamp>/report.json
lnt runtest --submit ~/myperfdb nt

# Run the Server
lnt runserver ~/myperfdb

# Connect in web browser
http://localhost:8000
LLVM Nightly Testing

LNT is a set of client and server tools for monitoring the performance of software over its lifecycle.

To setup your own LNT server or view the code visit the [main docs](#).

Data Sources

Several bots submit data to this LNT server:

Source

- Green Dragon
- LLVM Buildbots
### LNT-Broadwell-AVX2-O3__clang_DEV__x86_64 test results

<table>
<thead>
<tr>
<th>Run</th>
<th>Order</th>
<th>Start Time</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current</td>
<td>375349</td>
<td>10/19/2019 18:18:55</td>
<td>0:45:02</td>
</tr>
<tr>
<td>Previous</td>
<td>375348</td>
<td>10/19/2019 17:32:17</td>
<td>0:44:47</td>
</tr>
<tr>
<td>Baseline</td>
<td>309397</td>
<td>07/28/2017 15:59:11</td>
<td>0:42:23</td>
</tr>
</tbody>
</table>

### Tests Summary

<table>
<thead>
<tr>
<th>Status Group</th>
<th>#</th>
<th># (B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance Regressions</td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>Performance Improvements</td>
<td>0</td>
<td>19</td>
</tr>
<tr>
<td>Added Tests</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>Unchanged Tests</td>
<td>2560</td>
<td>2516</td>
</tr>
<tr>
<td>Total Tests</td>
<td>2560</td>
<td></td>
</tr>
</tbody>
</table>

**Runs:**
- 10/19/2019 18:18:55
- 10/19/2019 17:32:17
- 10/19/2019 16:33:37
- 10/19/2019 13:23:55

**Compare To:**
- 10/19/2019 18:18:55
- 10/19/2019
## Run-Over-Baseline Changes Detail

### Performance Regressions - execution_time

<table>
<thead>
<tr>
<th>Run</th>
<th>Δ (B)</th>
<th>Baseline</th>
<th>Current</th>
<th>σ (B)</th>
<th>Δ</th>
<th>σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>MultiSource/Benchmarks/FreeBench/fourinarow/fourinarow</td>
<td>245.00%</td>
<td>0.1855</td>
<td>1.0120</td>
<td>0.0642</td>
<td>-1.39%</td>
<td>0.0642</td>
</tr>
<tr>
<td>SingleSource/Benchmarks/Misc-C++-EH/spirit</td>
<td>129.00%</td>
<td>2.7478</td>
<td>6.2928</td>
<td>0.0994</td>
<td>-2.79%</td>
<td>0.0994</td>
</tr>
<tr>
<td>SingleSource/Benchmarks/Misc-C++/Large/ray</td>
<td>38.92%</td>
<td>1.7880</td>
<td>2.4839</td>
<td>0.0372</td>
<td>-0.03%</td>
<td>0.0372</td>
</tr>
<tr>
<td>SingleSource/Benchmarks/Misc/flops-1</td>
<td>19.69%</td>
<td>0.8919</td>
<td>1.0675</td>
<td>0.0130</td>
<td>0.16%</td>
<td>0.0130</td>
</tr>
<tr>
<td>SingleSource/Benchmarks/BenchmarkGame/fannkuch</td>
<td>18.98%</td>
<td>2.4340</td>
<td>2.8959</td>
<td>0.0652</td>
<td>1.43%</td>
<td>0.0652</td>
</tr>
<tr>
<td>SingleSource/Benchmarks/Misc/courafft</td>
<td>15.80%</td>
<td>2.9097</td>
<td>3.3695</td>
<td>0.0529</td>
<td>-0.82%</td>
<td>0.0529</td>
</tr>
<tr>
<td>SingleSource/Benchmarks/Shotout-C++/sieve</td>
<td>14.75%</td>
<td>1.5358</td>
<td>1.7624</td>
<td>0.0214</td>
<td>0.50%</td>
<td>0.0214</td>
</tr>
<tr>
<td>MultiSource/Benchmarks/MiBench/consumer-lame/consumer-lame</td>
<td>6.14%</td>
<td>0.1775</td>
<td>0.1884</td>
<td>0.0015</td>
<td>-1.62%</td>
<td>0.0015</td>
</tr>
<tr>
<td>MultiSource/Benchmarks/TSVC/Packing-flt/Packing-flt</td>
<td>4.07%</td>
<td>2.5481</td>
<td>2.6517</td>
<td>0.0269</td>
<td>-0.54%</td>
<td>0.0269</td>
</tr>
<tr>
<td>MultiSource/Applications/minisat/minisat</td>
<td>3.98%</td>
<td>5.2015</td>
<td>5.4087</td>
<td>0.0613</td>
<td>-0.51%</td>
<td>0.0613</td>
</tr>
<tr>
<td>MultiSource/Benchmarks/TSVC/Reductions-flt/Reductions-flt</td>
<td>3.78%</td>
<td>5.5050</td>
<td>5.7132</td>
<td>0.0316</td>
<td>0.37%</td>
<td>0.0316</td>
</tr>
</tbody>
</table>

### Compare To:

- 10/19/2019 18:18:55
- 10/19/2019 13:23:55
BUILD BOTS
Welcome to the Buildbot for the LLVM project!

Our goal is to provide extensive build and test coverage for all supported platforms.

If you are willing to donate some CPU cycles and some room on a hard drive, you are welcome. The instructions of how to add a build slave to the LLVM Buildbot infrastructure can be found in the How to Add Your Build Configuration to the LLVM Buildbot Infrastructure document.

Navigate:

- The Waterfall Display will give you a time-oriented summary of recent buildbot activity. Waterfall Help.
- The Grid Display will give you a developer-oriented summary of recent buildbot activity.
- The Transposed Grid Display presents the same information as the grid, but lists the revisions down the side.
- The Console presents a user-oriented status page.
- The Builders and their most recent builds are here.
- Recent Builds are summarized here, one per line.
- Buildslave information
- Changesource information.
- About this Buildbot
BUILD BOT

Console View

Categories: aosp clang clang.exp clang_fast libcxx libunwind lld lldb llvm openmp polly rev_iter sanitizer toolchain

Legend: Passed Failed Failed Again Running Exception Offline No data

Refine check for `__LIBCPP_C_HAS_NO_GETS` on FreeBSD
Summary:
In D67316 we added `__LIBCPP_C_HAS_NO_GETS` to signal that the C library does not provide `gets()`, and added a test for FreeBSD 13 or higher,
CONTRIBUTING

Tests

- Fix a bug and include regression tests
  - Visit https://bugs.llvm.org and search beginner

- Additional Tests would be great!
  - http://llvm.org/docs/Proposals/TestSuite.html
CONTRIBUTING

Structural

Plenty of room for growth in the LLVM test suite.

There is interest in adding support for

– Fortran
– OpenMP
– MPI
– Fixed support for other compilers
POINTERS

https://llvm.org/docs/TestingGuide.html
https://github.com/google/googletest/blob/master/googletest/docs/primer.md
https://llvm.org/docs/CommandGuide/FileCheck.html
https://llvm.org/docs/CommandGuide/lit.html
https://github.com/google/benchmark
http://llvm.org/docs/Lnt/index.html
http://lab.llvm.org:8011/
https://llvm.org/docs/HowToAddABuilder.html
https://bugs.llvm.org
ACKNOWLEDGEMENTS

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering, and early testbed platforms, in support of the nation’s exascale computing imperative.