LLVM Meetup in Munich

LLVM LIT Tutorial?

Alexey Sachkov
Agenda

- Disclaimers and background
- LIT test suites configuration
  - Easier and most probably known things
- Custom LIT test formats
  - More advanced and likely lesser-known things
Disclaimer

- Presented info is a personal experience/knowledge
  - It may not reflect best known methods!

- No info how to bootstrap LIT tests for a new project
  - More about how to configure them once you have them
Background

- A lot of headers to test

- End-to-end testing is a must have

- Variety of environments

```cpp
sycl::queue q;
sycl::range r{1024};
sycl::buffer<int> bufA(r);

q.submit([&](sycl::handler &cgh) {
    auto acc = bufA.get_access(
        cgh);
    cgh.parallel_for(r, [=](sycl::id<1> it) {
        // this is offloaded to an accelerator
        acc[it] = it + 42;
    });
});
```
LIT Test suites

And their configuration
LIT Test Suite

A directory containing `lit.cfg.py` or `lit.site.cfg.py`

Documentation: [Test Suites](#)

Config file is loaded and executed.

```
$ find . -name "lit.cfg.py"
./clang/test/lit.cfg.py
./clang/test/Unit/lit.cfg.py
./libcxxabi/test/lit.cfg.py
./lld/test/lit.cfg.py
./lld/test/Unit/lit.cfg.py
./bolt/test/lit.cfg.py
./bolt/test/Unit/lit.cfg.py
./libclc/test/lit.cfg.py
```
LIT Test Suite: config file

Name and format

```python
config.name = "My fancy test suite"

config.test_format = lit.formats.ShTest()
# Another available format is GoogleTest
```
LIT Test Suite: config file

Suffixes and excludes

```python
config.suffixes = [".c", ".cpp", ".my-custom-ext"]

config.excludes = [
    "Inputs",
    "still-work-in-progress",
    "example.cpp",
]

include_heavy_tests = bool(lit_config.params.get("INCLUDE_HEAVY_TEST", False))
if not include_heavy_tests:
    config.excludes.append("heavy-tests")
```
LIT Test Suite: config file

Suffixes and excludes: examples

llvm/llvm-project: `clang/test/lit.cfg.py`

intel/llvm: `sycl/test/lit.cfg.py`

Different suffixes are used based on LIT arguments
LIT Test Suite: config file

Substitutions

```python
cfg = config.substitutions.append(("%fancy_flags", "-O3 -fno-inline-functions -ffancy"))

tool_dirs = [os.path.join("path", "to", "some", "directory")]
tools = ["toolA", "toolB"]
# will add "toolA" substitution, *not* a "%toolA"
lit.llvm.llvm_config.add_tool_substitutions(tools, tool_dirs)

lit.llvm.llvm_config.use_default_substitutions()
lit.llvm.llvm_config.use_clang()
```
LIT Test Suite: config file

Substitutions: examples

llvm/llvm-project: llvm/test/lit.cfg.py
Also uses ToolSubst and FindTool helpers from lit.llvm.subst
LIT Test Suite: config file

Environment

config.environment["MY_FANCY_ENV"] = "true"

# Propagate existing env variables
lit.llvm.llvm_config.with_system_environment(["HOME", "INCLUDE"])

lit.llvm.llvm_config.with_environment("PATH", os.path.join("extra", "path"),
    append_path=True)

lit.llvm.llvm_config.with_environment("PATH", os.path.join("extra", "path-2"),
    append_path=True)
Available features

# Used by XFAIL, REQUIRES and UNSUPPORTED directives
config.available_features.add("my-fancy-feature")

if platform.system() not in ["Windows"]:
    config.available_features.add("not-a-windows")

if os.environ["SOMETHING"] == "something-else":
    config.available_features.add("feature-1")
LIT Test Suite: config file

Available features: examples

llvm/llvm-project: `clang/test/lit.cfg.py`
llvm/llvm-project: `llvm/test/lit.cfg.py`

intel/llvm: `sycl/test-e2e/lit.cfg.py`

- Adds features depending on which tools are present on the system
- Adds features based on output of tools launched by config
LIT local config files

```python
config.unsupported_features += ['feature_A', 'feature_B']

config.required_features += ['feature_C']
```

test-suite/

```
lit.cfg.py
test-a.cpp
test-b.cpp
category-1/
test-1a.cpp
test-1.bcpp
category-2/
lit.local.cfg.py
test-2a.cpp
test-2b.cpp
```
LIT Test formats
LIT Test Suite: config file

Name and format

```python
config.name = "My fancy test suite"

config.test_format = lit.formats.ShTest()

# Another available format is GoogleTest
```
LIT Test formats

class CustomTestFormat:
    # given a path to a concrete file,
    # need to produce lit.Test.Test object
    def getTestsForPath(...):
        pass

    # given a path to a directory, find all
    # tests in it
    def getTestsInDirectory(...):
        pass

    # execute a single test
    def execute(...):
        pass
LIT Test formats

Why? And what can you do with them?

- Add custom directives
- Generate “RUN” scripts on the fly
- Generate tests on the fly
- Integrate LIT with some testing framework
LIT Test formats

The test is hardcoded to GPU.

How to extend the test to other devices?

// RUN: clang++ -fsycl %s -o %t.out
// RUN: %t.out

sycl::queue q(sycl::gpu_device_v);
sycl::range r{1024};

q.parallel_for(r, [=](sycl::id<1> it) {
    // do something on device
});
LIT Test formats

Now the test is more generic, but it has 3 extra lines.

```
// RUN: clang++ -fsycl %s -o %t.out
// RUN: %CPU_RUN_PREFIX %t.out
// RUN: %GPU_RUN_PREFIX %t.out
// RUN: %ACC_RUN_PREFIX %t.out

sycl::queue q;
sycl::range r{1024};

q.parallel_for(r, [=](sycl::id<1> it) {
    // do something on device
});
```
LIT Test formats

Final solution: custom substitutions!

`%{run}` expands to several commands based on available devices

```cpp
// RUN: %{build} -o %t.out
// RUN: %{run} %t.out

sycl::queue q;
sycl::range r{1024};

q.parallel_for(r, [=](sycl::id<1> it) {
    // do something on device
});
```
LIT Test formats

We can specify environment requirements through custom directives

```cpp
// RUN: %{build} -o %t.out
// RUN: %{run} %t.out
// REQUIRES-INTEL-DRIVER: lin: 27501,
win: 101.4943

sycl::queue q;
sycl::range r{1024};

q.parallel_for(r, [=](sycl::id<1> it) {
    // do something on device
});
```
LIT Test formats

intellllvm: `sycl/test-e2e/format.py`

llvm/llvm-project: `llvm/utils/lit/lit/formats`
LIT Test formats: “virtual” tests

Background

SYCL 2020 spec says there is one header to rule them all

`sycl/sycl.hpp`

Does not scale well, but there is no understanding of how to split it

Let’s start by making sure that every header file is self-contained

How to test that?
LIT Test formats: “virtual” tests

class SYCLHeadersTest:
    def getTestsInDirectory(...):
        # here we extract a path to SYCL headers location from "config" object
        source_path = localConfig.sycl_include
        # and traverse the whole directory structure
        for dirpath, _, filenames in os.walk(source_path):
            for filename in filenames:
                # ignoring non-header files
                if not filename.endswith(".hpp"):
                    continue
                for t in self.getTestsForPath(..., os.path.join(dirpath, filename), ...):
                    yield t
LIT Test formats: “virtual” tests

class SYCLHeadersTest:
    ...

def execute(self, test, litConfig):
    # use path to header supplied during discovery step
    command = [test.config.clang, "-fsycl", "-fsyntax-only", "-include",
               test.file_path, "empty.cpp"]
    # execute command, handle output and exit code (simplified)
    out, err, exitCode = lit.util.executeCommand(command)

    status = lit.test.PASS if exitCode == 0 else lit.test.FAIL
    return lit.Test.Result(status, out + err)
LIT Test formats: “virtual” tests

intel/llvm: `sycl/test/format.py`
intel/llvm: `sycl/test/self-contained-headers`