Code Coverage Tool for Pytorch

Overview

This tool is designed for calculating code coverage for Pytorch project. It's an integrated tool. You can use this tool to run and generate both file-level and line-level report for C++ and Python tests. It will also be the tool we use in *CircleCI* to generate report for each main commit.

Simple

- *Simple command to run:*
  - `python oss_coverage.py`
- *Argument* `--clean` *will do all the messy clean up things for you*

But Powerful

- *Choose your own interested folder*:
  - Default folder will be good enough in most times
  - Flexible: you can specify one or more folder(s) that you are interested in
- *Run only the test you want:*
  - By default it will run all the c++ and python tests
  - Flexible: you can specify one or more test(s) that you want to run
- *Final report:*
  - File-Level: The coverage percentage for each file you are interested in
  - Line-Level: The coverage details for each line in each file you are interested in
  - Html-Report (only for `gcc`): The beautiful HTML report supported by `lcov`, combine file-level report and line-lever report into a graphical view.
- *More complex but flexible options:*
  - Use different stages like *--run, --export, --summary* to achieve more flexible functionality

How to use

This part will introduce about the arguments you can use when run this tool. The arguments are powerful, giving you full flexibility to do different work. We have two different compilers, `gcc` and `clang`, and this tool supports both. But it is recommended to use `gcc` because it's much faster and use less disk place. The examples will also be divided to two parts, for `gcc` and `clang`.

Preparation

The first step is to build *Pytorch* from source with `CODE_COVERAGE` option `ON`. You may also want to set `BUILD_TEST` option `ON` to get the test binaries. Besides, if you are under `gcc` compiler, to get accurate result, it is recommended to also select `CMAKE_BUILD_CONFIG=Debug`. See: how to adjust build options for reference. Following is one way to adjust build option:

```
# in build/ folder (all build artifacts must in `build/` folder)
cmake .. -DCODE_COVERAGE=ON -DBUILD_TEST=ON -DCMAKE_BUILD_CONFIG=Debug
```

Examples

The tool will auto-detect compiler type in your operating system, but if you are using another one, you need to specify it. Besides, if you are using `clang`, `llvm` tools are required. So the first step is to set some environment value if needed:

```
# set compiler type, the default is auto detected, you can check it at the start of
log.txt
export COMPILER_TYPE="CLANG"
# set llvm path for clang, by default is /usr/local/opt/llvm/bin
export LLVM_TOOL_PATH=...
```

Great, you are ready to run the code coverage tool for the first time! Start from the simple command:

```
python oss_coverage.py --run-only=atest
```

This command will run `atest` binary in `build/bin/` folder and generate reoports over the entire *Pytorch* folder. You can find the reports in `profile/summary`. But you may only be interested in the `aten` folder, in this case, try:

```
python oss_coverage.py --run-only=atest --interested-only=aten
```

In *Pytorch*, `c++` tests located in `build/bin/` and `python` tests located in `test/`. If you want to run `python` test, try:

```
python oss_coverage.py --run-only=test_complex.py
```

You may also want to specify more than one test or interested folder, in this case, try:

```
python oss_coverage.py --run-only=atest c10_logging_test --interested-only
aten/src/Aten c10/core
```

That it is! With these two simple options, you can customize many different functionality according to your need. By default, the tool will run all tests in `build/bin` folder (by running all executable binaries in it) and `test/` folder (by running `run_test.py`), and then collect coverage over the entire *Pytorch* folder. If this is what you want, try: *(Note: It's not recommended to run default all tests in clang, because it will take too much space)*

```
python oss_coverage.py
```

For more complex arguments and functionalities

GCC

The code coverage with `gcc` compiler can be divided into 3 step:

1. run the tests: `--run`
2. run `gcov` to get json report: `--export`
3. summarize it to human readable file report and line report: `--summary`

By default all steps will be run, but you can specify only run one of them. Following is some usage scenario:

1. Interested in different folder `—summary` is useful when you have different interested folder. For example,

```
# after run this command
python oss_coverage.py --run-only=atest --interested-folder=aten
# you may then want to learn atest's coverage over c10, instead of running the test
again, you can:
python oss_coverage.py --run-only=atest --interested-folder=c10 --summary
```

2. Run tests yourself When you are developing a new feature, you may first run the tests yourself to make sure the implementation is all right and then want to learn its coverage. But sometimes the test take very long time and you don't want to wait to run it again when doing code coverage. In this case, you can use these arguments to accerate your development (make sure you build pytorch with the coverage option!):

```
# run tests when you are devloping a new feature, assume the the test is `test_nn.py`
python oss_coverage.py --run-only=test_nn.py
# or you can run it yourself
cd test/ && python test_nn.py
# then you want to learn about code coverage, you can just run:
python oss_coverage.py --run-only=test_nn.py --export --summary
```

CLANG

The steps for `clang` is very similar to `gcc`, but the export stage is divided into two step:

1. run the tests: `--run`
2. run `gcov` to get json report: `--merge` `--export`
3. summarize it to human readable file report and line report: `--summary`

Therefore, just replace `--export` in `gcc` examples with `--merge` and `--export`, you will find it work!

Reference

For `gcc`

- See about how to invoke `gcov`, read Invoking gcov will be helpful

For `clang`

- If you are not familiar with the procedure of generating code coverage report by using `clang`, read Source-based Code Coverage will be helpful.