Homework 2: OpenTuner Tutorial

杨浩然 10195501441

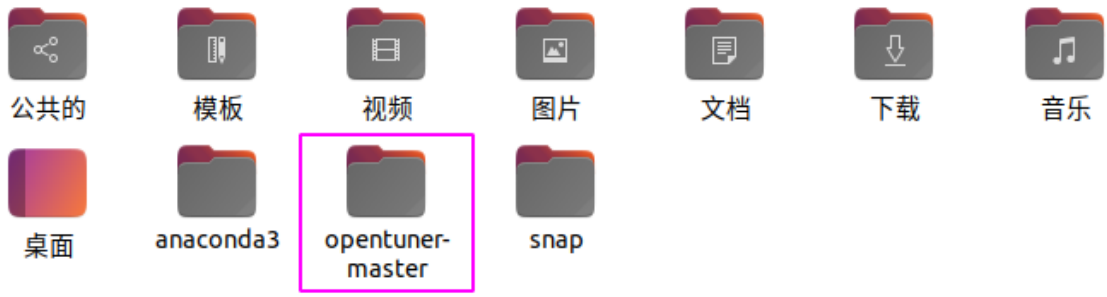
操作系统：ubuntu 20.04

2021-9-20

安装 OpenTuner

克隆仓库

将项目仓库 https://github.com/jansel/opentuner 克隆到本地



在 Anaconda 中安装

在 Anaconda 的 shell 中执行以下语句：

```
conda create --name=opentuner python=3.8
conda activate opentuner
pip install -r requirements.txt -r optional-requirements.txt
python setup.py develop
```

在执行 `pip install -r requirements.txt -r optional-requirements.txt` 过程中出现如下的错误：

```
ERROR: pip's dependency resolver does not currently take into account all the pa
ckages that are installed. This behaviour is the source of the following depende
ncy conflicts.
jill 0.10.1 requires requests, which is not installed.
```

应该是 jill 缺少了 requests，我手动安装 requests。

检查安装

To check an installation you can run tests:

```
pytest tests/*
```

```
(opentuner) younghojan@younghojan-XPS-15-7590:~/opentuner-master$ pytest tests/*
============================= test session starts ==============================
platform linux -- Python 3.8.12, pytest-6.2.5, py-1.10.0, pluggy-1.0.0
rootdir: /home/younghojan/opentuner-master
collected 23 items

tests/test_manipulator.py ...................                            [ 82%]
tests/test_technique.py ....                                             [100%]

============================== warnings summary ================================
../anaconda3/envs/opentuner/lib/python3.8/site-packages/future/standard_library/
__init__.py:65
  /home/younghojan/anaconda3/envs/opentuner/lib/python3.8/site-packages/future/s
tandard_library/__init__.py:65: DeprecationWarning: the imp module is deprecated
 in favour of importlib; see the module's documentation for alternative uses
    import imp

../anaconda3/envs/opentuner/lib/python3.8/site-packages/fn/iters.py:2
  /home/younghojan/anaconda3/envs/opentuner/lib/python3.8/site-packages/fn/iters
.py:2: DeprecationWarning: Using or importing the ABCs from 'collections' instea
d of from 'collections.abc' is deprecated since Python 3.3, and in 3.10 it will
stop working
    from collections import deque, Iterable

-- Docs: https://docs.pytest.org/en/stable/warnings.html
======================== 23 passed, 2 warnings in 0.28s ========================
```

为什么会有两个 warnings 捏?

- 第一个 warning 的意思是：`imp` 这个模块已经被弃用了，取代它是 `importlib`

  > Python将 **importlib** 作为标准库提供。它旨在提供 Python **import** 语法和(__import__() 函数)的实现。另外，**importlib** 提供了开发者可以创建自己的对象(即importer)来处理导入过程。
  >
  > 那么 **imp** 呢？还有一个 **imp** 模块提供了 **import** 语句接口，不过这个模块在 Python3.4 已经 **deprecated** 了。建议使用 **importlib** 来处理。

- 第二个 warning 的意思是：`from collections import abc` 这个用法被弃用了，取代它的是 `from collections.abc import YourModule`

Or run an example program:

```
./examples/rosenbrock/rosenbrock.py
```

run 了之后是这个样子：

```
(opentuner) younghojan@younghojan-XPS-15-7590:~/opentuner-master$ ./examples/ro
senbrock/rosenbrock.py
[     4s]     INFO opentuner.search.metatechniques: AUCBanditMetaTechniqueA: [('
NormalGreedyMutation', 157), ('UniformGreedyMutation', 116), ('RandomNelderMead
', 114), ('DifferentialEvolutionAlt', 114)]
[     8s]     INFO opentuner.search.metatechniques: AUCBanditMetaTechniqueA: [('
DifferentialEvolutionAlt', 126), ('UniformGreedyMutation', 126), ('RandomNelder
Mead', 125), ('NormalGreedyMutation', 124)]
[    10s]     INFO opentuner.search.plugin.DisplayPlugin: tests=1264, best {0: 8
.700294830684015, 1: 75.59874784865269}, cost time=60.2235, found by RandomNeld
erMead
[    12s]     INFO opentuner.search.metatechniques: AUCBanditMetaTechniqueA: [('
DifferentialEvolutionAlt', 126), ('NormalGreedyMutation', 125), ('RandomNelderM
ead', 125), ('UniformGreedyMutation', 125)]
[    17s]     INFO opentuner.search.metatechniques: AUCBanditMetaTechniqueA: [('
NormalGreedyMutation', 126), ('DifferentialEvolutionAlt', 126), ('RandomNelderM
ead', 125), ('UniformGreedyMutation', 124)]
[    20s]     INFO opentuner.search.plugin.DisplayPlugin: tests=2428, best {0: 1
```

阅读论文

论文写得很好，我非常有收获。

练习 Optimizing Block Matrix Multiplication 教程

Using OpenTuner

在 `examples/tutorials/` 中有一份 sample code `mmm_block.cpp` 和一个 autotuner 的 Python 程序 `mmm_tuner.py`。

Tuning Programs 都有一个普遍的结构：

- 首先是导入需要的库

```
from opentuner import MeasurementInterface
from opentuner import Result
```

- 创建一个 GccFlagsTuner 类的实例，这个实例可以通过 opentuner 中的 GccFlagsTuner 类来调整特定的参数。
`manipulator` 方法通过指定待调参数来定义 variable search space。

```
def manipulator(self):
  """
  Define the search space by creating a
  ConfigurationManipulator
  """
  manipulator = ConfigurationManipulator()
  manipulator.add_parameter(
    IntegerParameter('blockSize', 1, 10))
  return manipulator
```

- 在给定参数设置下，`run` 方法运行 opentuner 并且返回程序的性能。在这个示例中，待调参数 `blocksize` 是作为 compile-time(编译时) 常量输入的，程序每次运行时它都取一个在特定范围内的不同的值。

```
def run(self, desired_result, input, limit):
  """
  Compile and run a given configuration then
  return performance
  """
  cfg = desired_result.configuration.data

  gcc_cmd = 'g++ mmm_block.cpp '
  gcc_cmd += '-DBLOCK_SIZE='+ cfg['blockSize']
  gcc_cmd += ' -o ./tmp.bin'

  compile_result = self.call_program(gcc_cmd)
  assert compile_result['returncode'] == 0

  run_cmd = './tmp.bin'
```

```
  run_result = self.call_program(run_cmd)
  assert run_result['returncode'] == 0

  return Result(time=run_result['time'])
```

- 通过创建 `save_final_config()` 方法，我们可以显示 opentuner 的输出。这个方法保存一个 json 文件来存放 `blocksize` 的可取值。

```
def save_final_config(self, configuration):
  """called at the end of tuning"""
  print "Optimal block size written to mmm_final_config.json:",
configuration.data
  self.manipulator().save_to_file(configuration.data,
                                  'mmm_final_config.json')

if __name__ == '__main__':
  argparser = opentuner.default_argparser()
  GccFlagsTuner.main(argparser.parse_args())
```

运行 `mmm_tuner.py` 并查看结果：

```
python mmm_tuner.py --no-dups --stop-after=30
```

这是什么情况？并且最后输出的结果也和 tutorial 里不同。

尝试多次，最后的结果是 `Optimal block size written to mmm_final_config.json: {'BLOCK_SIZE': 10}`，有的时候会输出 `'BLOCK_SIZE': 9` 。

Creating Techniques

> This tutorial will walk through the basics of adding a new search technique to OpenTuner. We will add the technique pattern search.

这个部分的内容是将 Pattern Search 加到 OpenTuner 里面。

Code explanation

- ```
  def main_generator(self):
  ```

  `main_generator()` 是 SequentialSearchTechnique model 的核心函数，它会生成 opentuner.resultsdb.models.Configuration 对象。

- ```
      objective   = self.objective
  ```

  `objective` 对象是用于比较 configurations 的，它通常是 `MinimizeTime()` 的实例。

- ```
      driver      = self.driver
  ```

  `driver` 对象是用于和 results database 进行交互的，它可以用于查询该 technique 或其他 technique 所要求的配置的结果。

- ```
      manipulator = self.manipulator
  ```

  `manipulator` 对象允许 technique 更改和检查 configurations

- ```
      # start at a random position
      center = driver.get_configuration(manipulator.random())
  ```

  `manipulator.random()` 返回一个随机的初始 configuration。`driver.get_configuration` 将其转换为一个 `opentuner.resultdb.models.Configuration` 数据库中的记录（若此 configuration 是新的，则插入它，否则在数据库中进行查找）。此后，这个 configuration 就是不可变的了，并且将被分配一个 id 用于为其查找结果。

- ```
      yield center

        for param in manipulator.parameters(center.data):
  ```

  使用 manipulator 在 `opentuner.manipulator.Parameter` 中遍历。

懒得翻译了……看英文蛮好的……

- ```
          if param.is_primitive():
  ```

  For this initial version we will only handle `opentuner.manipulator.PrimitiveParameter` objects, which are based on `set_value`, `get_value`, and `legal_range` functions.

- ```
            # get current value of param, scaled to be in range [0.0, 1.0]
            unit_value = param.get_unit_value(center.data)
  ```

  We will use the convenience functions `get_unit_value` and `set_unit_value` which scale the parameter into a float from 0.0 to 1.0. `set_unit_value` will perform rounding for us if the underlying type is an integer.

- ```
            if unit_value > 0.0:
              # produce new config with param set step_size lower
              down_cfg = manipulator.copy(center.data)
  ```

  We copy `center.data` to get a mutable object to create our new configuration with. `down_cfg` is now a raw configuration, typically of type `dict`.

- ```
              param.set_unit_value(down_cfg, max(0.0, unit_value - step_size))
  ```

  Use the parameter to mutate `down_cfg` to have the value of param be `step_size` lower.

- ```
              down_cfg = driver.get_configuration(down_cfg)
              yield down_cfg
  ```

  Same as before, `driver.get_configuration` will convert the raw mutable configuration to a immutable `opentuner.resultdb.models.Configuration` database record which we can use to query for results. This then waits for results to be ready.

- ```
        #sort points by quality, best point will be points[0], worst is
  points[-1]
        points.sort(cmp=objective.compare)

        if objective.lt(points[0], center):
          # we found a better point, move there
          center = points[0]
        else:
          # no better point, shrink the pattern
          step_size /= 2.0
  ```

  Finally we use `objective.compare` and `objective.lt` to decide to either move the pattern or shrink the pattern depending on if we found a better configuration.

- ```
  # register our new technique in global list
  technique.register(BasicPatternSearch())
  ```

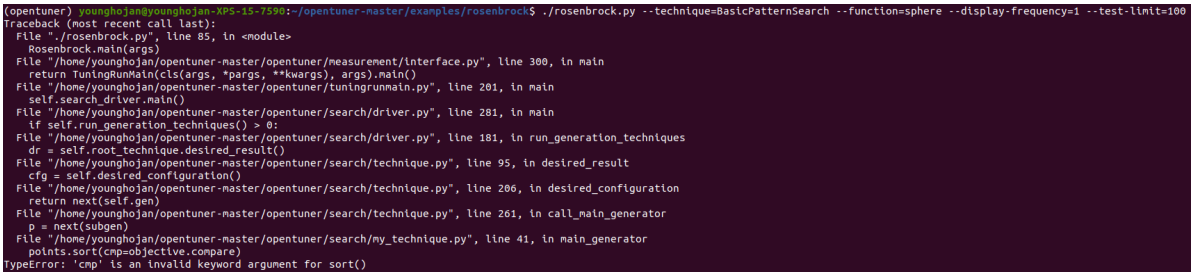
  This registers our technique in the global list to allow it to be selected with the `--technique=BasicPatternSearch` command line flag.

Running the new technique

在 `opentuner/examples/rosenbrock` 目录下运行 `rosenbrock.py` 。

```
./rosenbrock.py --technique=BasicPatternSearch --function=sphere --display-
frequency=1 --test-limit=100
```

Oops... 出错了


```

> TypeError: 'cmp' is an invalid keyword argument for sort()

在 Python2 中，`sort()` 函数是这样的：`sort(cmp=None, key=None, reverse=False)`

在 Python3 中，`sort()` 函数是这样的：`sort(*, key=None, reverse=None)`

`cmp` 参数没了....

在 https://github.com/jansel/opentuner 的 README.md 中，它要求我们在 Python=3.8 的 conda 虚拟环境中安装 OpenTuner：

Development installation

For development or running examples out of a git checkout, we recommend using miniconda3.

```
conda create --name=opentuner python=3.8
conda activate opentuner
pip install -r requirements.txt -r optional-requirements.txt
python setup.py develop
```

但是在 https://opentuner.org/tutorial/setup/ 的 Tutorial 中，它又是这样说明的：

System dependencies

A list of system dependencies can be found in debian-packages-deps which are primarily python 2.6+ (not 3.x) and sqlite3 (or your supported database backend of choice).

On Ubuntu/Debian there can be installed with:

```
sudo apt-get install `cat debian-packages-deps | tr '\n' ' '`
```

并且我发现其中不少代码都是依照 Python2 的语法标准编写的。

把这一句 `points.sort(cmp=objective.compare)` 改改好了。

在 `my_technique.py` 中作如下修改：

- 导入 `cmp_to_key` 以代替 `cmp`

```
from functools import cmp_to_key
```

- 用 `sorted()` 代替 `sort()`

```
        # points.sort(cmp=objective.compare)
        sorted(points, key=cmp_to_key(objective.compare))
```

再次运行：

```
(opentuner) younghojan@younghojan-XPS-15-7590:~/opentuner-master/examples/rosenbrock$ ./rosenbrock.py --technique=BasicPatternSearch --function=sphere --display-frequency=1 --test-limit=100
[     1s]    INFO opentuner.search.plugin.DisplayPlugin: tests=100, best {0: 84.46251223874833, 1: 651.6932403112371}, cost time=431837.9954, found by BasicPatternSearch
Final configuration {0: 84.46251223874833, 1: 651.6932403112371}
```

倒是能运行了，但为啥秒出结果？和 tutorial 又不太一样：

Running the new technique

Run the newly created technique, you need a program to tune. Some examples can be found in the `opentuner/examples` directory. The most simple is `opentuner/examples/rosenbrock` which implements a subset of the test functions described at http://en.wikipedia.org/wiki/Test_functions_for_optimization.

```
~/opentuner/examples/rosenbrock$ ./rosenbrock.py --technique=BasicPatte
[     1s]    INFO opentuner.search.plugin.DisplayPlugin: tests=17, best
[     2s]    INFO opentuner.search.plugin.DisplayPlugin: tests=42, best
[     3s]    INFO opentuner.search.plugin.DisplayPlugin: tests=67, best
[     4s]    INFO opentuner.search.plugin.DisplayPlugin: tests=92, best
[     5s]    INFO opentuner.search.plugin.DisplayPlugin: tests=101, bes
~/opentuner/examples/rosenbrock$
```

Improved version

把 PatternSearch 的地方做一些提升。There are three main changes:

- Yield_nonblocking for parallelism

```
...
self.yield_nonblocking(center)
...
        self.yield_nonblocking(down_cfg)
...
        self.yield_nonblocking(up_cfg)
...
  yield None # wait for all results
```

  This change allows tests to run in parallel. `self.yield_nonblocking(cfg)` requests that cfg be tested, but does not wait for the results. `yield None` waits for all prior `yield_nonblocking` result requests, and must be done before the configurations are compared against each other with `objective`.

- Support ComplexParameters

```
      else: # ComplexParameter
        for mutate_function in param.manipulators(center.data):
          cfg = manipulator.copy(center.data)
          mutate_function(cfg)
          cfg = driver.get_configuration(cfg)
          self.yield_nonblocking(cfg)
          points.append(cfg)
```

  This change adds support for ComplexParamers which do not have a linear value, but instead have a set of opaque manipulator functions. We simply add one point to points for each manipulator function.

- Sharing information with other techniques

```
    if (objective.lt(driver.best_result.configuration, center)
        and driver.best_result.configuration != points[0]):
      # another technique found a new global best, switch to that
      center = driver.best_result.configuration
```

> Since we may want to run this technique with other techniques, this change makes use of progress made by other techniques. If another technique found a better configuration, we switch to that new global best and abandon the current position.

再 run 一下试试（这一份代码依然用的是 `cmp`，还是得改改）：

```
(opentuner) younghojan@younghojan-XPS-15-7590:~/opentuner-master/examples/rosenbrock$ ./rosenbrock.py --technique=PatternSearch --function=sphere --display-frequency=1 --test-limit=100
[     1s]    INFO opentuner.search.plugin.DisplayPlugin: tests=103, best {0: 0.0019246377599984044, 1: 0.008878659268702904}, cost time=0.0001, found by PatternSearch
Final configuration {0: 0.0019246377599984044, 1: 0.008878659268702904}
```

跟没有 improve 过的 PatternSearch 得出的 configuration 确实不一样。