# Project 2

说明：本次实验的优化器是基于opentuner进行扩展，直接在opentuner的源码中进行的修改，主要是添加了autotuner.py以及在opentuner/search/中实现了两种算法，此外有涉及到driver.py manipulator.py 等的修改和添加。

代码在hw2

## 利用opentuner进行参数比较并测试

### 代码实现

```
def manipulator(self):
  """
  Define the search space by creating a
  ConfigurationManipulator
  """
  manipulator = ConfigurationManipulator()
  # manipulator.add_parameter(
  #   IntegerParameter('blockSize', 8,16))
  manipulator.add_parameter(
    IntegerParameter('opt_level', 0,3))
  manipulator.add_parameter(
    EnumParameter('blockSize',['8','16','32','64','128'])
  )
  return manipulator
```

设置需要比较的optimize level 0到3中的整数，设置block size,为['8','16','32','64','128']

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

  gcc_cmd = 'g++ Matrix_Multiplication.cpp '
  gcc_cmd += '-DBLOCK_SIZE='+ str(cfg['blockSize'])
  gcc_cmd += ' -O{0}'.format(cfg['opt_level'])
  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'])
```

使用opentuner测试Matrix_Multiplication.cpp最优的optimize level和blocksize组合

### 测试结果

```
(opentuner) wangwenqingdeMacBook-Pro:hw2 wangwenqing$ python autotuner.py --no-dups --stop-a
fter=30
[     2s] WARNING opentuner.search.driver: Result callback 4614 (requestor=DifferentialEvolu
tionAlt) pending for 1 generations 1 results available
[     7s]    INFO opentuner.search.metatechniques: AUCBanditMetaTechniqueA: [('DifferentialE
volutionAlt', 482), ('UniformGreedyMutation', 9), ('NormalGreedyMutation', 8), ('RandomNelde
rMead', 2)]
[    28s]    INFO opentuner.search.metatechniques: AUCBanditMetaTechniqueA: [('DifferentialE
volutionAlt', 493), ('NormalGreedyMutation', 4), ('UniformGreedyMutation', 4)]
[    35s]    INFO opentuner.search.metatechniques: AUCBanditMetaTechniqueA: [('DifferentialE
volutionAlt', 493), ('NormalGreedyMutation', 4), ('UniformGreedyMutation', 4)]
Optimal block size written to mmm_final_config.json: {'opt_level': 3, 'blockSize': '32'}
(opentuner) wangwenqingdeMacBook-Pro:hw2 wangwenqing$
```

# 两种搜索算法的实现和结果比较

## 一、搜索算法设置

在opentuner文件夹中的opentuner/search/technique.py中看到如下代码

```
argparser = argparse.ArgumentParser(add_help=False)
argparser.add_argument('--technique', '-t', action='append', mzhan, 8 years ago: • in-place parameter operator (1)
                       help="which technique to use")
argparser.add_argument('--list-techniques', '-lt', action='store_true',
                       help="list techniques available and exit")
argparser.add_argument('--generate-bandit-technique', '-gbt', action='store_true',
                       help="randomly generate a bandit to use")
```

所以先运行-lt查看可以使用的搜索算法，结果如下

```
(opentuner) wangwenqingdeMacBook-Pro:hw2 wangwenqing$ python autotuner.py --no-dups --stop-a
fter=30 -lt
PureRandom
ga-OX3
ga-OX1
ga-PX
ga-CX
ga-PMX
ga-base
UniformGreedyMutation05
UniformGreedyMutation10
UniformGreedyMutation20
NormalGreedyMutation05
NormalGreedyMutation10
NormalGreedyMutation20
DifferentialEvolution
DifferentialEvolutionAlt
DifferentialEvolution_20_100
RandomNelderMead
RegularNelderMead
RightNelderMead
MultiNelderMead
RandomTorczon
RegularTorczon
RightTorczon
```

例如我们尝试使用AUCBanditMetaTechniqueA进行搜索,其实现见opentuner/search/bandittechniques.py(https://github.com/jansel/opentuner/blob/2d23d76a9a15eb953e405c3263aa74ce9b8adbd2/opentuner/search/bandittechniques.py)

搜索结果如下

```
(opentuner) wangwenqingdeMacBook-Pro:hw2 wangwenqing$ python autotuner.py --no-dups --stop-a
fter=30 --technique=AUCBanditMetaTechniqueA --display-frequency=1 --test-limit=100
[     2s]    INFO opentuner.search.plugin.DisplayPlugin: tests=10, best {'opt_level': 2, 'bl
ockSize': '16'}, cost time=0.0059, found by DifferentialEvolutionAlt
[     2s]    INFO opentuner.search.plugin.DisplayPlugin: tests=17, best {'opt_level': 3, 'bl
ockSize': '128'}, cost time=0.0058, found by NormalGreedyMutation
Optimal block size written to mmm_final_config.json: {'opt_level': 3, 'blockSize': '128'}
(opentuner) wangwenqingdeMacBook-Pro:hw2 wangwenqing$
```

可以看到很快就搜索到了最优的组合，效果不错

## 二、Grid search

**1.实现过程**

主要实现在main_generator()函数中，main_generator() 是 SequentialSearchTechnique 模型的核心函数。

**几个主要的对象**：

Objective:（在 opentuner/opentuner/search/objective.py 中定义）用于使用用户定义的质量指标来比较配置。 它通常是 MinimizeTime() 的一个实例，它只查看结果对象所用时间。

Driver:（在 opentuner/opentuner/search/driver.py 中定义）用于与结果数据库交互。 它可用于查询此技术和其他技术请求的配置的结果。

Manipulator:（在 opentuner/opentuner/search/manipulator.py 中定义）允许该技术进行更改和检查配置。 从概念上讲，它是一个参数对象列表，这些对象要么是primitive，并且具有诸如 set_value / get_value / legal_range 之类的功能，要么具有一组将更改底层配置的不透明操纵器函数的复杂功能。这里我们用到的是primitive这块的功能

**定义搜索起点**

同时设置当前的最优为搜索起点，通过yield使生效

```
    inital={'opt_level':0,'blockSize':8}
    # start at a random position
    center = driver.get_configuration(inital)
    yield center
    # print("center type",type(center))
    best=center
    yield best
```

**定义搜索过程**

过程主要为如果当前的blocksize没有到最大的128，则进行blocksize翻倍，opt_level不变。如果blocksize达到128，则blocksize设为最小的8，并将opt_level加1。如果blocksize为128，opt_level为3，均为最大值则记录当前点并推出循环。

注意int str的一些转换

```
    while True:
      points = list()
      for param in manipulator.parameters(center.data):
        if param.is_primitive():
          print('center',center.data)
          if int(center.data['blockSize'])<128:
            next_cfg=manipulator.copy(center.data)
            blocksize_next=int(next_cfg['blockSize'])*2
            next_cfg['blockSize']=str(blocksize_next)
            unit_value = param.get_unit_value(next_cfg)
            next_cfg=driver.get_configuration(next_cfg)
            yield next_cfg
            points.append(next_cfg)
          elif int(center.data['opt_level'])<3:
            go_cfg=manipulator.copy(center.data)
            opt_level_next=int(go_cfg['opt_level'])+1
            go_cfg['opt_level']=opt_level_next
            go_cfg['blockSize']=str(8)
            go_cfg=driver.get_configuration(go_cfg)
            yield go_cfg
            points.append(go_cfg)

          else:
            points.append(center)
            sys.exit()
```

**更新center和best值**

对所有记录的point值调用objective中的compare进行排序,并更新center以及当前的best

```
      points.sort(key=cmp_to_key(objective.compare))
      center=points[0]
      if objective.lt(points[0], best):
        best = points[0]
```

**注册当前搜索算法**

在全局列表中注册当前搜索算法，以允许使用 --technique=GridSearch 命令行标志选择它。

```
technique.register(GridSearch())
```

**2.运行命令**

python autotuner.py --no-dups --stop-after=30 --technique=GridSearch --display-frequency=1

**3.运行结果**

打印出所有搜索到的组合，并按照时间1s间隔打印最好的组合

```
(opentuner) wangwenqingdeMacBook-Pro:hw2 wangwenqing$ python autotuner.py --no-dups --stop-a
fter=30 --technique=GridSearch --display-frequency=1
center {'opt_level': 0, 'blockSize': 8}
center {'opt_level': 0, 'blockSize': '16'}
center {'opt_level': 0, 'blockSize': '32'}
center {'opt_level': 0, 'blockSize': '64'}
center {'opt_level': 0, 'blockSize': '128'}
center {'opt_level': 1, 'blockSize': '8'}
center {'opt_level': 1, 'blockSize': '16'}
center {'opt_level': 1, 'blockSize': '32'}
center {'opt_level': 1, 'blockSize': '64'}
center {'opt_level': 1, 'blockSize': '128'}
center {'opt_level': 2, 'blockSize': '8'}
center {'opt_level': 2, 'blockSize': '16'}
center {'opt_level': 2, 'blockSize': '32'}
center {'opt_level': 2, 'blockSize': '64'}
center {'opt_level': 2, 'blockSize': '128'}
center {'opt_level': 3, 'blockSize': '8'}
center {'opt_level': 3, 'blockSize': '16'}
center {'opt_level': 3, 'blockSize': '32'}
[     1s]    INFO opentuner.search.plugin.DisplayPlugin: tests=19, best {'opt_level': 0, 'bl
ockSize': '32'}, cost time=0.0057, found by GridSearch
```

**4.分析不同组合耗时不同的原因**

```
{'opt_level': 0, 'blockSize': '16'}, cost time=0.0063, found by GridSearch
```

```
{'opt_level': 1, 'blockSize': '8'}, cost time=0.0058, found by GridSearch
```

```
{'opt_level': 1, 'blockSize': '32'}, cost time=0.0059, found by GridSearch
```

```
{'opt_level': 1, 'blockSize': '128'}, cost time=0.0060, found by GridSearch
```

```
{'opt_level': 2, 'blockSize': '16'}, cost time=0.0057, found by GridSearch
```

```
{'opt_level': 2, 'blockSize': '64'}, cost time=0.0059, found by GridSearch
```

```
{'opt_level': 3, 'blockSize': '16'}, cost time=0.0059, found by GridSearch
```

可以看到不同的组合性能存在差异，尝试分析其原因为

当分块过大时,数据无法充分利用缓存或寄存器空间进行重用,就有可能被置换出去,从而导致缓存未命中的情况频繁发生;当 分块过小时,可能会造成较大的调度成本,无法充分发挥分块带来的优势,最优和最差的分块大小可能导致性能相差数倍.

O0不做优化。O1对程序做部分编译优化。当设置O2选项时，编译器并不进行循环打开loop unrolling以及函数内联。与O1比较而言，O2优化增加了编译时间的基础上，提高了生成代码的执行效率。 O3在O2的基础上进一步优化。

当blocksize未达到最优（不大不小）时，优化等级对于程序性能的影响可能并不明显。

## 三、自主实现的搜索算法 pattern search

**1.实现过程**

实现一个不包含并发的pattern search。

初始步长设置为0.2。由于这里对所有的点进行了归一化处理。

使用`opentuner.manipulator.PrimitiveParameter`对象, 其主要的功能函数有`set_value`, `get_value`, and `legal_range`等 .

在get_unit_value中获取一个已经归一化的值。set_unit_value中会首先assert值是否在0到1之间，并置一个归一化后的值。相关实现在manipulator.py中。

进行向上或者向下的搜索。首先复制 center.data 以获得一个可变对象来创建我们的新的 configuration。使用参数改变 down_cfg ， 在 param 的值上减去 step_size。

和以前一样，driver.get_configuration 将原始可变配置转换为不可变 opentuner.resultdb.models.Configuration 数据库记录。

```
      for param in manipulator.parameters(center.data):
        if param.is_primitive():
          unit_value = param.get_unit_value(center.data)

          if unit_value > 0.0:
            down_cfg = manipulator.copy(center.data)
            param.set_unit_value(down_cfg, max(0.0, unit_value - step_size))
            down_cfg = driver.get_configuration(down_cfg)
            yield down_cfg
            points.append(down_cfg)

          if unit_value < 1.0:
```

```
            up_cfg = manipulator.copy(center.data)
            param.set_unit_value(up_cfg, min(1.0, unit_value + step_size))
            up_cfg = driver.get_configuration(up_cfg)
            yield up_cfg
            points.append(up_cfg)
```

使用objective的compare和lt来决定是否移动到新加入的点继续搜索。如果新加入的点和之前的点相比，不是最优，则将步长减半

```
      points.sort(key=cmp_to_key(objective.compare))
      if objective.lt(points[0], center):
        center = points[0]
      else:
        step_size /= 2.0
```

**2.运行命令**

```
python autotuner.py --no-dups --stop-after=30 --technique=BasicPatternSearch --display-frequency=1 --test-limit=100
```

**3.运行结果**

搜索中

```
[     2s]    INFO opentuner.search.plugin.DisplayPlugin: tests=1, best {'opt_level': 3, 'blockSize': '64'}, cost time=0.0070, found by BasicPatternSearch
[     3s]    INFO opentuner.search.plugin.DisplayPlugin: tests=2, best {'opt_level': 0, 'blockSize': '128'}, cost time=0.0063, found by BasicPatternSearch
[     5s]    INFO opentuner.search.plugin.DisplayPlugin: tests=3, best {'opt_level': 2, 'blockSize': '8'}, cost time=0.0061, found by BasicPatternSearch
[     6s]    INFO opentuner.search.plugin.DisplayPlugin: tests=8, best {'opt_level': 0, 'blockSize': '128'}, cost time=0.0058, found by BasicPatternSearch
[    31s]    INFO opentuner.search.plugin.DisplayPlugin: tests=12, best {'opt_level': 0, 'blockSize': '32'}, cost time=0.0057, found by BasicPatternSearch
[    31s]    INFO opentuner.search.plugin.DisplayPlugin: tests=12, best {'opt_level': 0, 'blockSize': '32'}, cost time=0.0057, found by BasicPatternSearch
Optimal block size written to mmm_final_config.json: {'opt_level': 0, 'blockSize': '32'}
(opentuner) wangwenqingdeMacBook-Pro:hw2 wangwenqing$
```

## 四、两种算法的优劣

Grid search

是一种暴力搜索的方法。

优：穷尽所有组合，总能在一定时间中找到最优的组合。在总组合数较少时搜索速度较快

劣：对于组合总数更多的情况，可能grid search的搜索效果就并不是特别理想。

pattern search

优：可以首先确定大致最优的所在位置，然后再通过缩小步长精确搜索，适用总组合数较多的情况

劣：虽然算法是收敛的，但是对于组合总数较少的情况，搜索效率可能会略低于grid search

# reference


https://opentuner.org/tutorial/

https://towardsdatascience.com/grid-search-in-python-from-scratch-hyperparameter-tuning-3cca8443727b