计算机体系结构 lab3

PB20111704 张宇昂

NMRU实现方法

实验文档中前4步只是简单的`Ctrl-C`&`Ctrl-V`，在此不进行展示（**最关键的是不要忘记在`Sim_object`中添加相关项，容易忘记**），我们直接从设计NMRU模块的部分进行介绍，在`nmru_rp.cc`文件中，除了修改前缀`LRU`为`NMRU`外，我们要做的是设计`getVictim`函数：

第一版设计的`NMRU`函数如下，先找到MRU对应的`candidate`，然后随机选取一个非MRU的`candidate`即可，先用和`mru_rp.cc`中`getVictim`相同的实现方法找到对应的`candidate`，然后再用一个while循环随机选取`candidate`，保证和MRU选取的不同即可：

```
ReplaceableEntry*
NMRU::getVictim(const ReplacementCandidates& candidates) const
{
    // There must be at least one replacement candidate
    assert(candidates.size() > 0);

    std::shared_ptr<NMRUReplData> mru_replacement_data =
        std::static_pointer_cast<NMRUReplData>(candidates[0]->replacementData);

    // Get MRU lastTouchTick
    for (const auto& candidate : candidates) {
        std::shared_ptr<NMRUReplData> candidate_replacement_data =
        std::static_pointer_cast<NMRUReplData>(candidate->replacementData);

        // Stop searching entry if a cache line that doesn't warm up is found.
        if (candidate_replacement_data->lastTouchTick == 0) {
            return candidate;
        } else if (candidate_replacement_data->lastTouchTick >
                        mru_replacement_data->lastTouchTick) {

            mru_replacement_data =
                std::static_pointer_cast<NMRUReplData>(candidate-
>replacementData);
        }
    }
    // Choose one candidate at random and make sure its not MRU
    ReplaceableEntry* victim;
    std::shared_ptr<NMRUReplData> victim_replacement_data;
    do{
        victim = candidates[random_mt.random<unsigned>(0, candidates.size() -
1)];
        victim_replacement_data =
            std::static_pointer_cast<NMRUReplData>(victim->replacementData);

    }while(victim_replacement_data->lastTouchTick == mru_replacement_data-
>lastTouchTick);

    return victim;
```

```
}
```

但是实际的替换效率并不好，我们考虑对该过程进行优化：

```
ReplaceableEntry*
NMRU::getVictim(const ReplacementCandidates& candidates) const
{
    // There must be at least one replacement candidate
    assert(candidates.size() > 0);

    std::shared_ptr<NMRUReplData> mru_replacement_data =
        std::static_pointer_cast<NMRUReplData>(candidates[0]->replacementData);

    // Get MRU lastTouchTick
    for (const auto& candidate : candidates) {
        std::shared_ptr<NMRUReplData> candidate_replacement_data =
        std::static_pointer_cast<NMRUReplData>(candidate->replacementData);

        // Stop searching entry if a cache line that doesn't warm up is found.
        if (candidate_replacement_data->lastTouchTick == 0) {
            return candidate;
        } else if (candidate_replacement_data->lastTouchTick >
                        mru_replacement_data->lastTouchTick) {

            mru_replacement_data =
                std::static_pointer_cast<NMRUReplData>(candidate-
>replacementData);
        }
    }
    // Choose one candidate at random and make sure its not MRU
    ReplaceableEntry* victim;
    std::shared_ptr<NMRUReplData> victim_replacement_data;
    do{
        victim = candidates[random_mt.random<unsigned>(0, candidates.size() -
1)];
        victim_replacement_data =
            std::static_pointer_cast<NMRUReplData>(victim->replacementData);

    }while(victim_replacement_data->lastTouchTick == mru_replacement_data-
>lastTouchTick);

    return victim;
}
```

修改配置

1. 在 `configs/common/ObjectList.py` 添加：

```
repl_list = ObjectList(getattr(m5.objects,'BaseReplacementPolicy',None))
```

2. 在`two_level.py`中加入assoc、tag_latency以及replacement_policy的设置，需要注意的是这个部分需要在cache连接到cpu前进行设置否则会出现问题（也可能是我当时其他地方设置有错误）：

```
# Set tag latency and assoc
system.cpu.dcache.tag_latency = args.tag_latency
system.cpu.dcache.assoc = args.l1d_assoc

# Set the replacement policy
if args.replacement_policy == "Random":
    system.cpu.dcache.replacement_policy = RandomRP()
    system.l2cache.replacement_policy = RandomRP()
elif args.replacement_policy == "NMRU":
    system.cpu.dcache.replacement_policy = NMRURP()
    system.l2cache.replacment_policy = NMRURP()
elif args.replacement_policy == "LIP":
    system.cpu.dcache.replacement_policy = LIPRP()
    system.l2cache.replacement_policy = LIPRP()
elif args.replacement_polict == "LRU":
    system.cpu.dcache.replacement_policy = LRURP()
    system.l2cache.replacement_policy = LRURP()
```

模拟结果

脚本文件如下：

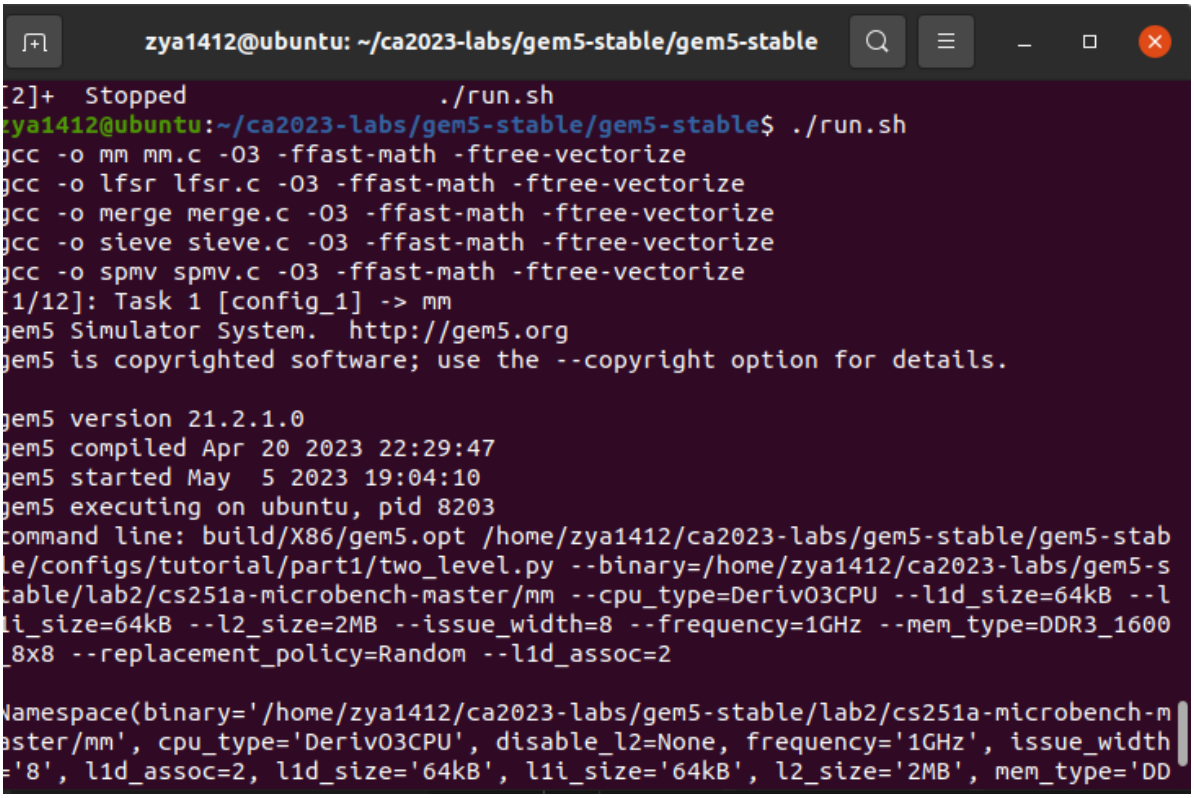
```
#!/bin/bash

GEM5_BASE=~/ca2023-labs/gem5-stable/gem5-stable
SRC_DIR=~/ca2023-labs/gem5-stable/lab2/cs251a-microbench-master
RESULT_DIR=~/ca2023-labs/gem5-stable/lab3/result
TARGET=~/ca2023-labs/gem5-stable/gem5-stable/configs/tutorial/part1/two_level.py
# TARGET=~/ca2023-labs/gem5-stable/gem5-stable/configs/example/se.py

NUM=1

# Clean result dir
rm -rf ${RESULT_DIR}/*
for FILE in mm; do
    mkdir -p ${RESULT_DIR}/${FILE}
done

for FILE in mm; do
# for FILE in mm; do

    cd $SRC_DIR && make clean --silent && make all && cd $GEM5_BASE

    # Random with 2 assoc
    echo [${NUM}/15]: Task 1 [config_1] -\> $FILE
    build/X86/gem5.opt ${TARGET} --binary=${SRC_DIR}/${FILE} --
cpu_type='DerivO3CPU' \
        --l1d_size='64kB' --l1i_size='64kB' --l2_size='2MB' --issue_width=8 \
        --frequency='1GHz' --mem_type='DDR3_1600_8x8' --
replacement_policy='Random' \
```

```
        --l1d_assoc=2
    cp -r m5out/ ${RESULT_DIR}/${FILE}/config_1/
    ((NUM++))

    # Random with 4 assoc
    echo [${NUM}/15]: Task 2 [config_2] -\> $FILE
    build/X86/gem5.opt ${TARGET} --binary=${SRC_DIR}/${FILE} --
cpu_type='DerivO3CPU' \
        --l1d_size='64kB' --l1i_size='64kB' --l2_size='2MB' --issue_width=8\
        --frequency='1GHz' --mem_type='DDR3_1600_8x8' --
replacement_policy='Random' \
        --l1d_assoc=4
    cp -r m5out/ ${RESULT_DIR}/${FILE}/config_2/
    ((NUM++))

    # Random with 8 assoc
    echo [${NUM}/15]: Task 3 [config 3] -\> $FILE
    build/X86/gem5.opt ${TARGET} --binary=${SRC_DIR}/${FILE} --
cpu_type='DerivO3CPU' \
        --l1d_size='64kB' --l1i_size='64kB' --l2_size='2MB' --issue_width=8\
        --frequency='1GHz' --mem_type='DDR3_1600_8x8' --
replacement_policy='Random' \
        --l1d_assoc=8
    cp -r m5out/ ${RESULT_DIR}/${FILE}/config_3/
    ((NUM++))

    # NMRU with 2 assoc
    echo [${NUM}/15]: Task 4 [config 4] -\> $FILE
    build/X86/gem5.opt ${TARGET} --binary=${SRC_DIR}/${FILE} --
cpu_type='DerivO3CPU' \
        --l1d_size='64kB' --l1i_size='64kB' --l2_size='2MB' --issue_width=8\
        --frequency='1GHz' --mem_type='DDR3_1600_8x8' --
replacement_policy='NMRU' \
        --l1d_assoc=2
    cp -r m5out/ ${RESULT_DIR}/${FILE}/config_4/
    ((NUM++))

    # NMRU with 4 assoc
    echo [${NUM}/15]: Task 5 [config 5] -\> $FILE
    build/X86/gem5.opt ${TARGET} --binary=${SRC_DIR}/${FILE} --
cpu_type='DerivO3CPU' \
        --l1d_size='64kB' --l1i_size='64kB' --l2_size='2MB' --issue_width=8\
        --frequency='1GHz' --mem_type='DDR3_1600_8x8' --
replacement_policy='NMRU' \
        --l1d_assoc=4
    cp -r m5out/ ${RESULT_DIR}/${FILE}/config_5/
    ((NUM++))

    # NMRU with 8 assoc
    echo [${NUM}/15]: Task 6 [config 6] -\> $FILE
    build/X86/gem5.opt ${TARGET} --binary=${SRC_DIR}/${FILE} --
cpu_type='DerivO3CPU' \
        --l1d_size='64kB' --l1i_size='64kB' --l2_size='2MB' --issue_width=8\
        --frequency='1GHz' --mem_type='DDR3_1600_8x8' --
replacement_policy='NMRU' \
```

```
        --l1d_assoc=8
    cp -r m5out/ ${RESULT_DIR}/${FILE}/config_6/
    ((NUM++))

    # LIP with 2 assoc
    echo [${NUM}/15]: Task 7 [config 7] -\> $FILE
    build/X86/gem5.opt ${TARGET} --binary=${SRC_DIR}/${FILE} --
cpu_type='DerivO3CPU' \
        --l1d_size='64kB' --l1i_size='64kB' --l2_size='2MB' --issue_width=8\
        --frequency='1GHz' --mem_type='DDR3_1600_8x8' --replacement_policy='LIP'
\
        --l1d_assoc=2
    cp -r m5out/ ${RESULT_DIR}/${FILE}/config_7
    ((NUM++))

    # LIP with 4 assoc
    echo [${NUM}/15]: Task 8 [config 8] -\> $FILE
    build/X86/gem5.opt ${TARGET} --binary=${SRC_DIR}/${FILE} --
cpu_type='DerivO3CPU' \
        --l1d_size='64kB' --l1i_size='64kB' --l2_size='2MB' --issue_width=8\
        --frequency='1GHz' --mem_type='DDR3_1600_8x8' --replacement_policy='LIP'
\
        --l1d_assoc=4
    cp -r m5out/ ${RESULT_DIR}/${FILE}/config_8
    ((NUM++))

    # LIP with 8 assoc
    echo [${NUM}/15]: Task 9 [config 9] -\> $FILE
    build/X86/gem5.opt ${TARGET} --binary=${SRC_DIR}/${FILE} --
cpu_type='DerivO3CPU' \
        --l1d_size='64kB' --l1i_size='64kB' --l2_size='2MB' --issue_width=8\
        --frequency='1GHz' --mem_type='DDR3_1600_8x8' --replacement_policy='LIP'
\
        --l1d_assoc=8
    cp -r m5out/ ${RESULT_DIR}/${FILE}/config_9
    ((NUM++))

    # LRU with 2 assoc
    echo [${NUM}/15]: Task 10 [config 10] -\> $FILE
    build/X86/gem5.opt ${TARGET} --binary=${SRC_DIR}/${FILE} --
cpu_type='DerivO3CPU' \
        --l1d_size='64kB' --l1i_size='64kB' --l2_size='2MB' --issue_width=8\
        --frequency='1GHz' --mem_type='DDR3_1600_8x8' --replacement_policy='LRU'
\
        --l1d_assoc=2
    cp -r m5out/ ${RESULT_DIR}/${FILE}/config_10
    ((NUM++))

    # LRU with 4 assoc
    echo [${NUM}/15]: Task 11 [config 11] -\> $FILE
    build/X86/gem5.opt ${TARGET} --binary=${SRC_DIR}/${FILE} --
cpu_type='DerivO3CPU' \
        --l1d_size='64kB' --l1i_size='64kB' --l2_size='2MB' --issue_width=8\
        --frequency='1GHz' --mem_type='DDR3_1600_8x8' --replacement_policy='LRU'
\
```

```
        --l1d_assoc=4
    cp -r m5out/ ${RESULT_DIR}/${FILE}/config_11
    ((NUM++))

    # LRU with 8 assoc
    echo [${NUM}/15]: Task 15 [config 15] -\> $FILE
    build/X86/gem5.opt ${TARGET} --binary=${SRC_DIR}/${FILE} --
cpu_type='DerivO3CPU' \
        --l1d_size='64kB' --l1i_size='64kB' --l2_size='2MB' --issue_width=8\
        --frequency='1GHz' --mem_type='DDR3_1600_8x8' --replacement_policy='LRU'
\
        --l1d_assoc=8
    cp -r m5out/ ${RESULT_DIR}/${FILE}/config_15
    ((NUM++))

    # q2 config 1
    echo [${NUM}/15]: Task 13 [config 13] -\> $FILE
    build/X86/gem5.opt ${TARGET} --binary=${SRC_DIR}/${FILE} --
cpu_type='DerivO3CPU' \
        --l1d_size='64kB' --l1i_size='64kB' --l2_size='2MB' --issue_width=8\
        --frequency='2.2GHz' --mem_type='DDR3_1600_8x8' --
replacement_policy='Random' \
        --l1d_assoc=16 --tag_latency=3
    cp -r m5out/ ${RESULT_DIR}/${FILE}/config_2_1
    ((NUM++))

    # q2 config 2
    echo [${NUM}/15]: Task 14 [config 14] -\> $FILE
    build/X86/gem5.opt ${TARGET} --binary=${SRC_DIR}/${FILE} --
cpu_type='DerivO3CPU' \
        --l1d_size='64kB' --l1i_size='64kB' --l2_size='2MB' --issue_width=8\
        --frequency='2.2GHz' --mem_type='DDR3_1600_8x8' --
replacement_policy='NMRU' \
        --l1d_assoc=8 --tag_latency=4
    cp -r m5out/ ${RESULT_DIR}/${FILE}/config_2_2
    ((NUM++))

    # q2 config 2
    echo [${NUM}/15]: Task 15 [config 15] -\> $FILE
    build/X86/gem5.opt ${TARGET} --binary=${SRC_DIR}/${FILE} --
cpu_type='DerivO3CPU' \
        --l1d_size='64kB' --l1i_size='64kB' --l2_size='2MB' --issue_width=8\
        --frequency='2.2GHz' --mem_type='DDR3_1600_8x8' --
replacement_policy='LIP' \
        --l1d_assoc=8 --tag_latency=5
    cp -r m5out/ ${RESULT_DIR}/${FILE}/config_2_3
    ((NUM++))
done
```

运行截图如下：



性能分析列表:

replacement policy	simTicks	assoc	miss rate
Random	4006964000	2	0.051682
Random	4006964000	4	0.058148
Random	4006327000	8	0.059839
NMRU (version 1)	4006851000	2	0.056848
NMRU (version 1)	4006964000	4	0.059636
NMRU (version 1)	4006964000	8	0.060134
LIP	4006327000	2	0.047836
LIP	4006327000	4	0.067426
LIP	4007395000	8	0.104036
LRU	4006327000	2	0.047887
LRU	4006327000	4	0.066626

replacement policy	simTicks	assoc	miss rate
LRU	4006327000	8	0.102502
NMRU (version 2)	4006327000	2	0.047887
NMRU (version 2)	4006327000	4	0.053948
NMRU (version 2)	4006088000	8	0.057203

结果分析：

- 可以看到NMRU的效率与实现方法有关
- 性能最好的是NMRU（version 2）
- 大部分实现的simTick随assoc的增大减小，missrate随assoc增大而增大，这是因为因为对于测试使用的mm.c，大部分时间按顺序访问内存，所有分支都是数据独立的，使得NMRU命中率提高，执行时间更短

第二种实际情况下的结果对比：

case	simTicks	missrate
case 1(Random 16 assoc 100ps Lookup time)	1909410230	0.060474
case 2(NMRU 8 assoc 500ps Lookup time)	1909520795	0.057353
case 3(LIP 8 assoc 550ps Lookup time)	1911719810	0.104098

综合simTicks、missrate以及assoc考虑，我认为NMRU是更优的替换策略