Programming Homework 5 for CA

Due 2023 Jan 2nd

You should do this assignment on your own.

If you have any problems, please contact the teaching assistant in time.

In this assignment, you will learn about the Cache replacement policy. You will implement one or two Cache replacement policies and compare them with existing policies, and consider tradeoffs in cache design.

Step 1

In GEM5, the default cache replacement policy used is LRU, in this step, we will give you a simple tutorial about how to create a new replacement policy.

If we want to add a new replacement policy, we need encapsulate it in a SimObject. So first let's copy one of the existing policies. You can find existing policies in /src/mem/cache/replacement_policies/:

```
cp lru_rp.cc nmru_rp.cc
cp lru_rp.hh nmru_rp.hh
```

Within each file, rename any LRU specific prefixes to NMRU(lru to nmru).

Then, each SimObject has a Python class which is associated with it. This Python class describes the parameters of your SimObject that can be controlled from the Python configuration files. For our NMRU replacement policy, we are just going to inherit all of the parameters from the BaseReplacementPolicy. Thus, we simply need to declare a new class for our SimObject and set it's name and the C++ header that will define the C++ class for the SimObject.

We edit the file, ReplacementPolicies.py function , in function /src/mem/cache/replacement_policies:

```
#for example, Insert in line 85
class NMRURP(BaseReplacementPolicy):
  type = 'NMRURP'
  cxx_class = 'NMRURP'
  cxx_header = "mem/cache/replacement_policies/nmru_rp.hh"

```

And we need register the C++ file. Each SimObject must be registered with SCons so that its Params object and Python wrapper is created. Additionally, we also have to tell SCons which C++ files to compile. To do this, modify the SConscipt file in the directory that your SimObject is in. For each SimObject, add a call to SimObject and for each source file add a call to Source. In this example, you need to add the following to src/mem/cache/replacement_policies/SConscript:

```
Source('nmru_rp.cc')
```

And add 'NMRURP' in sim_objects of SimObject(also in src/mem/cache/replacement_policies/SConscript) in correct position.

Now, you should be able to compile the code. And you can add addtional paramters to use your own replacement policy.

Here we provide a way to make it configurable in configs/example/se.py. First, we need add some new options in configs/common/Options.py:

```
parser.add_argument("--l1d_repl", default="LRURP",
                    choices=ObjectList.rp_list.get_names(),
                    help = "replacement policy for l1")

parser.add_argument("--l2_repl",  default="LRURP",
                    choices=ObjectList.rp_list.get_names(),
                    help = "replacement policy for l2")
```

You can insert it in about line 212, after the argument "--list-rp-types". And connect the argument to the dcache, in configs/common/CacheConfig.py:

```
# This is a new function to add
def _get_repl(repl_option):
      if repl_option == None:
              return NULL

      replacement_policy = ObjectList.rp_list.get(repl_option)
      return replacement_policy()

# Add the code to the original function
def _get_cache_opts(level, options):
      ...

      repl_attr = '{}_repl'.format(level)
      if hasattr(options, repl_attr):
              opts['replacement_policy'] = _get_repl(getattr(options, repl_attr))

      ...
```

Now, when you use se.py, you can use "--list-rp-types" to check which strategies you can use, and use "l1d_repl=NMRURP"/"l2_repl=NMRURP" to customize the cache replacement policy.

If everything is right, you can find the changed replacement_policy type in [system.cpu.dcache.replacement_policy] and [system.l2.replacement_policy] in config.ini.

Step 2

In step1, we create a copied policy, NMRU(not most recently used). In this step, we need to truly implement it. You can try to understand the getVictim function in nmru_rp.cc(Which is in fact LRU's function), and understanding the RandomRP may help.

Here are some references that may be useful:

Replacement Policies of gem5

Creating your first SimObject

A nmru tutorial from github

(Optional) After implemented the NMRU, you can do similar steps to implement LIP(LRU Insertion Policy), which is come from there. You can refer the LRU and BIP policies which gem5 has implemented. (You will find that class LIPRP have been defined in ReplacementPolicies.py and inherited from BIPRP)

Step 3

In this step, we will do some architectural exploration, and we will use a micro benchmark, Bench.tar. Use

```
make gen_arr
make
```

to get the executable file for test.

Suppose you are designing the L1D and L2 data caches of a new processor, based on gem5's out-of-order O3CPU. For simplicity, you can choose from only three replacement policies for the caches: Random, NMRU, LIP. If you do not implement the LIP, you can choose FIFO and TreePLRU as substitute strategies.

Task 1

Run the simulations of your policies on the benchmark, and assume you have a 64KB L1D, and a 2MB L2. Assume both L1D and L2 policies have to be the same, and that you are using 8-way set associativity on both. Other parameters: 64KB L1I + LRU, 2GHz frequency, 8-issue OOO core.

Determine which policy is the best, and explain why.

Task 2

Later, the hardware implementation team is worried about hitting high frequencies with a complex policy. As the associativity increases, the cost for some policies rises, whereas the cost for the other policies stays the same. Thus, the implementation team suggests the following maximum associativities, and other capacity keep the same for all caches designs.

RP	Random	NMRU	LIP	FIFO	TreePLRU
Max assoc.	16	8	4	8	8

Answer the questions:

Which Policies performes better? Why?

Is the cache replacement/associativity important for this workload, or are you only getting benefits from clock cycle? Explain why the cache architecture is important/unimportant.

For those policies that you discard, think about the circumstances under which it would perform better than the policy your chosen.

What to Hand In

Turn in your assignment on Canvas

- All files below should be attached as a zip file. The submission name should contain the name and ID numbers of the student, for example: zhangsan_921104681912_HW4.zip.
- nmru_rp.cc/hh file you changed, and lip_rp.cc/hh if you do.
- A report for all the questions in step3 with necessary notes. You should list and analyze some data from stats.txt to support your answer. Recommended submission formats are doc, pdf and md.
- stats.txt and config.ini files for simulation of Bench/mm.c.
- If you changed something else, submit it and explain it in the report.

Some Hints

- You may need rm -rf build/ and recomplie the gem5 to solve some strange errors.
- Sometimes you may find quite a few differences in stats.txt. You can pick some data that you think is important and make an analysis them.
- Sometimes you may find that there are almost no differences in stats.txt. There may be two reasons for this: one, not all code in bench is cache sensitive and you can use mm.c as the most notable benchmark; two, it is normal for different replacement strategies to have similar performance here.
- The assignment is based on gem5 101 homework5 from gem5, cs259 hw2 and cs251a hw3 from UCLA.