Project 3 Instructions

This project is split into two components. First, you will read a short paper ("IPV Paper") about a special type of cache replacement policy. On Canvas, you will take a multiple-choice quiz on this reading to make sure that you have understood how the policy works. Then, using some parts of the policy learned in the paper, you will implement a new cache replacement policy in gem5.

NOTE: It is highly recommended to read the paper first, then take the quiz, before implementing the project in code. The quiz will help you understand what you need to know.

Objective: Understand the cache replacement functionality and implement a new cache policy in the gem5 simulator. There are four main functions invoked by each cache replacement policy: `reset`, `touch`, `invalidate`, and `getVictim`. You can find the different functions about cache replacement policies in the following website:

https://www.gem5.org/documentation/general_docs/memory_system/replacement_policies/

1. `reset()`: It is used to initialize a replacement data. It is called upon only insertion of a cache block (at the beginning or upon replacement).

2. `touch()`: It is invoked upon a cache hit. It should update the replacement data of the cache block.

3. `invalidate()`: It is called whenever an entry is invalidated. It should invalidate the replacement data, setting the entry as next probable victim for eviction.

4. `getVictim()`: It is called when there is a miss, and an eviction must be done. It searches among all replacement candidates for an entry to be evicted (based on the replacement data of candidates being searched).

Please understand the above functions carefully, in order to understand how a cache replacement policy is implemented. These functions are invoked by top-level cache module, e.g., with functions implemented through files tags/base_set_assoc.hh, tags/base.cc, and base.cc in the directory `src/mem/cache`. These files also define functions for a cache's structure and its management, including for set associative caches. Trying to locate above four functions (related to cache block replacement) in these files could help you to understand when and why they are invoked.

Understanding LRU Policy: In the design of set associative caches, an important design parameter to analyze is the block replacement policy. Block replacement policy essentially determines which block (out of all the blocks in a set) must be evicted from the cache when it is time to bring in new data. A popular and easy to understand cache replacement policy is Least Recently Used (LRU). In this scheme, we evict the block that has been unused for the longest period. While it seems to be one of the most intuitively promising block replacement schemes, implementation complexity is its main disadvantage. LRU must maintain an access history for each block, which slows down the cache. Thus, most caches implement only an approximation of LRU.

You will find implementation of various cache replacement policies in the file lru_rp.cc and lru_rp.hh, located in the directory `gem5/src/mem/cache/replacement_policies`. In LRUIPV policy, when a block needs to be promoted upon a hit, it may affect the replacement metadata of other blocks in the set. To implement such mechanism, you may need to share the replacement related data among all blocks within the same set. Then, it needs to be handled through a shared pointer to the object for the replacement data. You can refer to such management in existing implementations, e.g., tree_plru (where replacement data about a tree is shared among its nodes i.e., blocks within the set).

Resources:

- Least Recently Used (LRU) Replacement Policy
- Tree Pseudo-LRU

NOTE: Before starting the project, please make sure your modifications to source code or configuration of parameters made during projects 1 or 2 are reverted back. If not, please either discard them or obtain a new copy of the gem5 again, and then rebuild the gem5.

Folder Setup: You will be generating a total of 3 different benchmark outputs. Just like Project 2, there is only 1 problem split into various subtasks. For each of 3 configurations, you will run a benchmark which outputs the files "config.json", "config.ini", and "stats.txt". At the end, you will generate a patch file which is a portable file that makes it easy to apply your gem5 modifications in project 3 to other unmodified gem5 projects. We will use this for grading to ensure that your implementation works. To generate the described file structure, run the following command as a single command (not 3 separate commands):

```
mkdir Project3-Submission Project3-Submission/Problem1 Project3-
Submission/Problem1/conf1 Project3-Submission/Problem1/conf2
Project3Submission/Problem1/conf3
```

Problem 1 - Implementing LRU-IPV

In this project, you will need to implement LRU-IPV policy. It is defined in the paper "*Insertion and Promotion for Tree-Based PseudoLRU Last-Level Caches*" available on Canvas. You need to understand Sections 1-4 and Figures 2 and 3 in the paper. Then, implement the transition graph with IPV [0 0 1 0 3 0 1 2 1 0 5 1 0 0 1 11 13] (same as the paper) for LRU management.

Task 1: Define your replacement policy in ReplacementPolicies.py

Introduce a new class LRUIPVRP() by adding the following lines to the 'ReplacementPolicies.py' file, found in the `gem5/src/mem/cache/replacement_policies` directory.

```
class LRUIPVRP(BaseReplacementPolicy):
       type = 'LRUIPVRP'
       cxx_class = 'LRUIPVRP'
       cxx_header = "mem/cache/replacement_policies/lru_ipv.hh"
       numWays = Param.Int(Parent.assoc, "Set associativity")
```

Task 2: Change in SConscript

Add the following line at the end of the file 'SConscript' in the same directory:

```
Source('lru_ipv.cc')
```

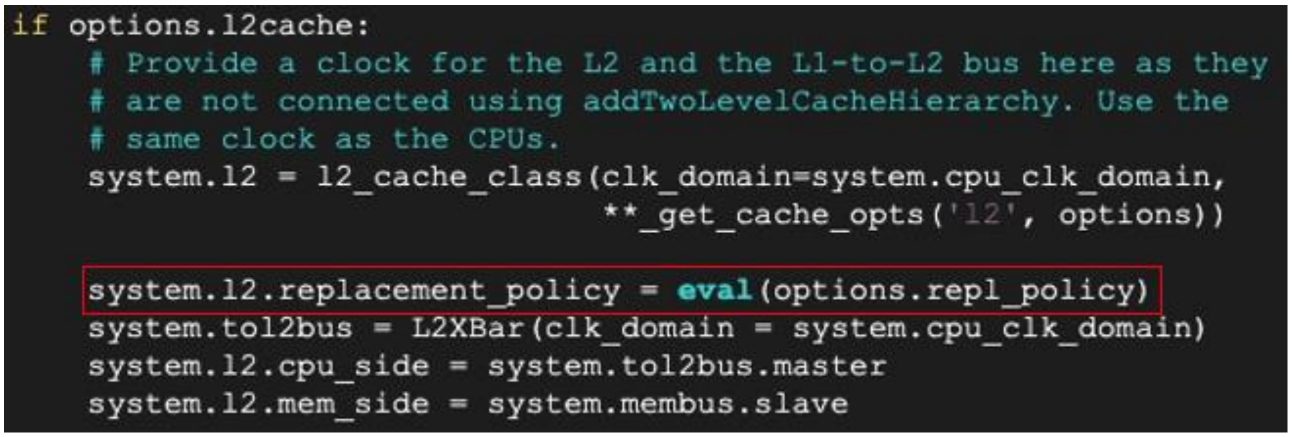
Task 3: Add repl_policy flag to gem5 options for command-line management

The default configuration for the option/flag for replacement policy is LRURP(), which is also gem5's default choice for all caches. Open `gem5/configs/common/Options.py` and add the following code under "Cache Options":

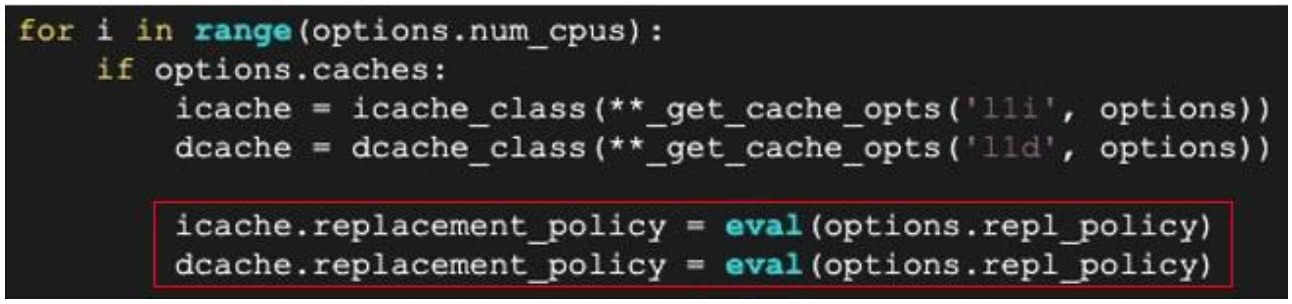
```
parser.add_option("--repl_policy", type="string", default="LRURP()")
```

Task 4: Set replacement policies of caches as per command-line option

We will set the same replacement policy of all caches (L1 data and instruction caches, L2 cache, etc.), based on the value of the command-line option. We can provide this information by modifying the information about a cache-based memory hierarchy, which is defined in `gem5/configs/common/CacheConfig.py`. You can add the code boxed in red in the figures below.


```
if options.l2cache:
    # Provide a clock for the L2 and the L1-to-L2 bus here as they
    # are not connected using addTwoLevelCacheHierarchy. Use the
    # same clock as the CPUs.
    system.l2 = l2_cache_class(clk_domain=system.cpu_clk_domain,
                               **_get_cache_opts('l2', options))

    system.l2.replacement_policy = eval(options.repl_policy)
    system.tol2bus = L2XBar(clk_domain = system.cpu_clk_domain)
    system.l2.cpu_side = system.tol2bus.master
    system.l2.mem_side = system.membus.slave
```


```
for i in range(options.num_cpus):
    if options.caches:
        icache = icache_class(**_get_cache_opts('l1i', options))
        dcache = dcache_class(**_get_cache_opts('l1d', options))

        icache.replacement_policy = eval(options.repl_policy)
        dcache.replacement_policy = eval(options.repl_policy)
```

Task 5: Implement LRU-IPV Cache Functionality in Code

Create two files named lru_ipv.cc and lru_ipv.hh in the directory of `gem5/src/mem/cache/replacement_policies`. You will need to implement the four main cache functions: `reset`, `touch`, `invalidate` and `getVictim`, in addition to any other functions that you need to implement the cache policy. You may hard-code the IPV vector in either the lru_ipv.hh or lru_ipv.cc file.

NOTE: You may notice that many of the things that you read about in the IPV paper need to be implemented in code in your implementation. You may also notice that some of the things you read about in the IPV paper do not need to be implemented in code.

Once you make the above modifications to fully implement the LRU-IPV Cache, rebuild gem5 by running the following command:

```
scons build/ARM/gem5.opt
```

In order to test the validity of your implementation, view the "Testing LRUIPV Implementation" document on Canvas, which walks through the debugging and validation process.

Task 6: Running Benchmarks on your Implementation

Once you are confident that your implementation is correct, you will need to run the benchmarks for the three different configurations just like in Project 2. However, this time because the parameters of the configurations are only different in the benchmarks, you will not need to rebuild before modifying each configuration. Instead, you will just need to change the parameters in the benchmarks accordingly. Then, run the dijkstra_small benchmark for all three configurations of LRU-IPV given in the table. You don't need to run any other benchmarks. Change the following parameters in the benchmark command below to observe 3 different configurations:

Configuration	L1 cache	L2 cache	L2 associativity	Benchmark
1	32 kB data cache, 32 kB instruction cache	**128 kB**	16	dijkstra_small
2	32 kB data cache, 32 kB instruction cache	**256 kB**	16	dijkstra_small
3	32 kB data cache, 32 kB instruction cache	**512 kB**	16	dijkstra_small

You can use --repl_policy flag to specify the replacement policy. For this project, you will run the 'dijkstra_small' benchmark on the three configurations given in the table. Both the configurations need to be evaluated only for LRUIPV replacement policy, i.e., --repl_policy="LRUIPVRP()". When running the benchmarks for the different configurations, you will be changing the parameters in the benchmarks command as highlighted in the code blocks below. In order to run the '**dijkstra_small**' benchmark for each configuration, run the following benchmark command (as a single line):

```
./build/ARM/gem5.opt -d [path to target directory]
./configs/example/se.py --cpu-type=DerivO3CPU --caches --l1i_size=32kB -
-l1d_size=32kB --l2cache --l2_assoc=16 --repl_policy="LRUIPVRP()" --
l2_size=128kB -c ./benchmarks/dijkstra/dijkstra_small -o
./benchmarks/dijkstra/input.dat
```

For each configuration, you must store the results of each benchmark in the appropriate directory, which is done by putting the path to the appropriate directory for the configuration as the parameter "path to target directory" in the above commands. You created these directories as the first command in the project instructions.

Creating the Patch File

Just like in Project 2, for this project you will create a patch file, which allows the auto grader to run your implementation of the code on an independent gem5 build. In order to create a correct patch file, please download a new gem5.zip file from the Project1 assignment on Canvas. Unzip the file, change its name to 'gem5_clean' and put it in the same directory as your current gem5 directory. Then run the following command outside the gem5 directory:

```
diff -ruN gem5_clean/src/cpu/pred/ gem5/src/cpu/pred/ > project3.patch
```

NOTE: Make sure the "N" in -ruN is uppercase. You can open project3.patch and locate all your modifications to SConscript, ReplacementPolicies.py, CacheConfig.py, Options.py, lru_ipv.hh and lru_ipv.cc with a "+" sign at the beginning of each line modified, which is automatically added by the above command.

Then, move project3.patch into the Project3-Submission directory. Then you should see project3.patch and the directory Problem1.

Submission Instructions

Submit 3 files to canvas:

1) Submit the file lru_ipv.cc.
2) Submit the file lru_ipv.hh.
3) Submit 'LastName-Project3-Submission.zip. This contains the directory structure outlined in the "Project Deliverables" section. You should be zipping just the folder "Project3-Submission" and then submitting the resulting zip file entitled "LastName-Project3-Submission.zip" where 'LastName' is the last name of the group member submitting the file to Canvas.

Only one group member should make the submission through Canvas. All team members in the group will receive the same score. Please follow the naming conventions correctly so that the auto grader will correctly grade your assignment.

Project Deliverable Structure

1. The project directory structure that you build should contain the following:
    a. A parent directory: Project3-Submission.
    b. 1 file inside Project3-Submission: project3.patch.
    c. 1 directory inside Project3-Submission: Problem1.
    d. Problem1 should contain 3 directories: conf1, conf2, and conf3.

e. Each of these configuration directories should contain config.ini, config.json, and stats.txt, all generated by gem5 after running the benchmark command of **your assigned benchmark**. The directory structure should be as follows:

```
Project3-Submission
     |
     +---project3.patch
     |
     +---Problem1
         +---conf1
         |   +---config.ini
         |   +---config.json
         |   +---stats.txt
         |
         +---conf2
         |   +---config.ini
         |   +---config.json
         |   +---stats.txt
         |
         +---conf3
             +---config.ini
             +---config.json
             +---stats.txt
```

Rubric

Project 3 [10 points]

a) Accuracy: correct values in the stats.txt file. **[3 points]**

b) Useful comments and clean code: Your implementations of lru_ipv.cc and lru_ipv.hh should contain useful and descriptive comments for each function and variable that you implement. **[2 points]**

c) Quiz: a 10-question multiple choice quiz on LRU implementations and concepts from the IPV paper on Canvas. 2 hours for each quiz attempt. Only 3 attempts. The maximum score across all attempts is taken. **[5 points]**