Lab 1 Report for Computer Architecture

Bohan Cui
ID: 221220089
NJU, Department of computer science and technology
Nanjing, China
bohan.cui@outlook.com

I. Run

It is an **push-button** solution for this lab.

You can just set the environment variable GEM5_ROOT to your gem5 root directory, and run [make run] in the lab code directory.

About the detailed instruction of setup environment and install prerequisite, please refer to the README.

II. Core Implementation

Actually, There is nothing much to say about the Gem5 Scripts. Only very slight modification is made to the original code, as below.

```
parser.add_argument("--clock", default='1GHz', nargs='?', type=
    str, help="Clock rate of the CPU.")
parser.add_argument("--cache_line_size", default='64', nargs='?',
    type=str, help="Cache line size. Default: 64B.")
```

Listing 1. Modification to the original code

Then I will explain the *Makefile* structure, for the automation mainly relies on the Makefile.

```
mkdir -p $(TARGET1)
@for clock in $(CLOCK_RATES); do \
    echo "\033[33m Run with clk: $$clock, L1ICache: 16kB, L1DCache: 64kB, L2Cache: 256kB, CacheLine: 64B \033[0m"; \
    $(GEM5_ROOT)/build/X86/gem5.opt --outdir=./m5out $(SCRIPT) \
        --binary ./Eratosthenes \
        --clock $$clock; \
    $(MAKE) copy_results TARGETN=1; \
    python3 $(RETRIEVE) $(TARGET1)/result_clock.json $(TARGET1)/stats.txt $$clock "simSeconds"; \
    python3 $(RETRIEVE) $(TARGET1)/result_cycles.json $(TARGET1)/stats.txt $$clock "system.cpu.numCycles"; \
    python3 $(RETRIEVE) $(TARGET1)/result_insts.json $(TARGET1)/stats.txt $$clock "simInsts"; \
    $(MAKE) rename_results TARGET_NAME=$$clock TARGETN=1; \
done
```

Listing 2. Makefile Slice

This is two of the six parts (p1,p2) of the tasks, as an example. It will call *lab1.py* and pass the needed parameters to it.

```
for line in gem5_output_lines:
    if line.strip().startswith(target_entry):
        parts = line.split()
        if len(parts) >= 2:
            target_value = parts[1]
            break
if target_value is None:
    print(f"Error: fail to find '{target_entry}'!")
    return
print(f"\033[34mCaught:{target_value}\033[0m")
```

Listing 3. retrieve.py Slice

Then it will copy the results to the new directory, and call *retrieve.py* to get the data I need from the stats.txt. The retrieved data will be stored into a json file, for example, result_clock.json.

And it will rename the results to the unique independent variable, for example, stats -¿ 1GHz.txt.

After all the parts are done, it will call *plot.py* to generate the plots.

```
def miss_rates_vs_line_size(miss_rate_file, output_dir, title, ylabel):
    with open(miss_rate_file, 'r') as f:
        miss_rate_data = json.load(f)
    line_sizes = [parse_log_size_value(k) for k in miss_rate_data.keys()]
    miss_rates = [float(v) * 100 for v in miss_rate_data.values()]
    plt.figure(figsize=(10, 6))
    plt.plot(line_sizes, miss_rates, marker='o')
    plt.title(title)
    plt.xlabel('Line Size (log2(kB))')
    plt.ylabel(ylabel)
    plt.grid(True)
    os.makedirs(output_dir, exist_ok=True)
    output_file = os.path.join(output_dir, f"{title.lower().replace(' ','_')}.png")
    plt.savefig(output_file)
    plt.close()
```

Listing 4. plot.py Slice

To make the plot more balance, I adopt the logarithm of the cache size.

III. Results

I only paste the results graphs here. The analysis is **in the Extended Questions section**. You can see the detailed results in the reserved sample_res folder or the result folder after running it.

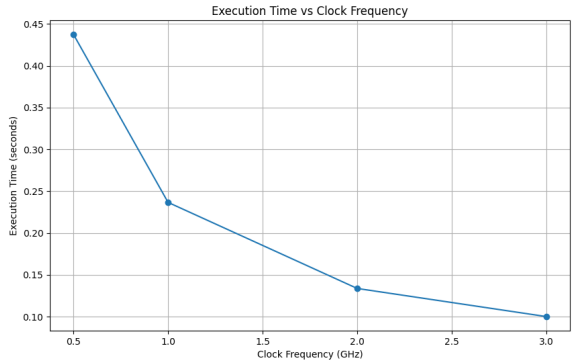

Fig. 1. Exe time vs Clock rate

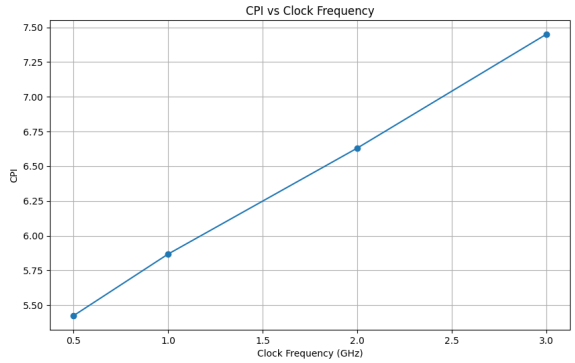

Fig. 2. CPI vs Clock rate

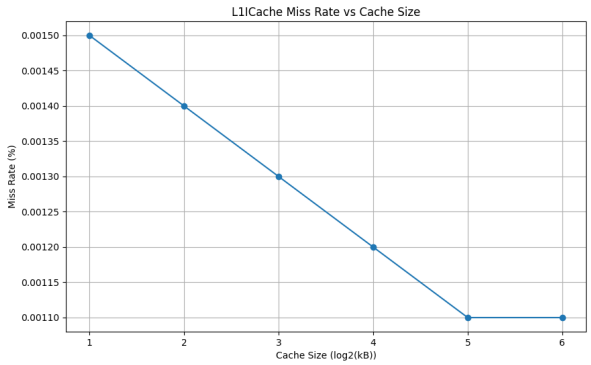

Fig. 3. L1I Cache Miss Rate vs Cache size

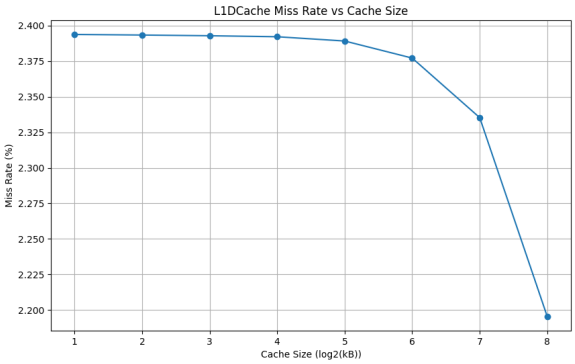

Fig. 4. L1D Cache Miss Rate vs Cache size

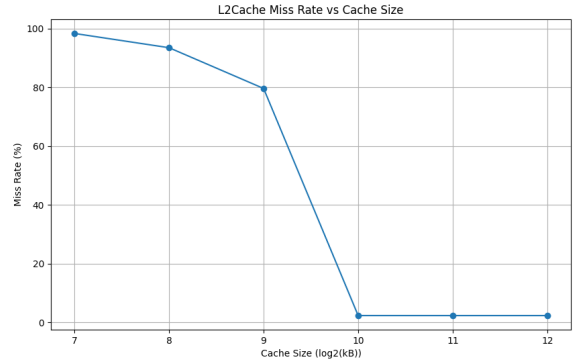

Fig. 5. L2 Cache Miss Rate vs Cache size

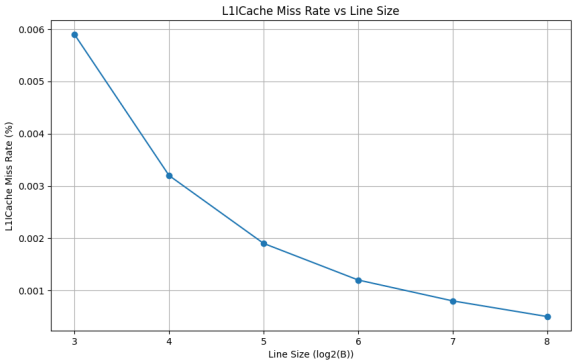

Fig. 6. L1I Cache Miss Rate vs Line size

IV. Extended Questions(Analysis)

A. Q1

Q: How is the running time related to the CPU clock rate, why?

A: It is obvious that basically the running time is inversely proportional to the CPU clock rate, as shown in Fig.1. This is because the the cycle of all the behaviors of this system is supposed to stay the same. Then when the clock rate f becomes kf, the running time should become $\frac{1}{k}$ of the original time.

However, we can observe that the cycles will increase a bit, as below. (Meanwhile, it will cause the CPI increase a bit, as in Fig.2.)

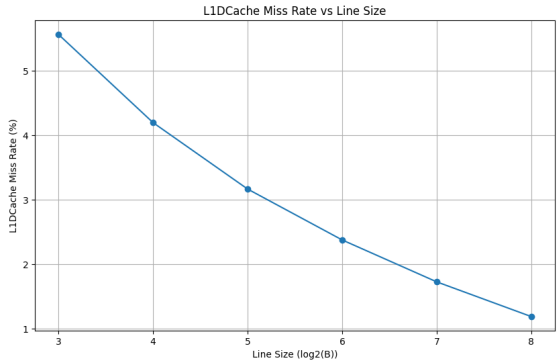

Fig. 7. L1D Cache Miss Rate vs Line size

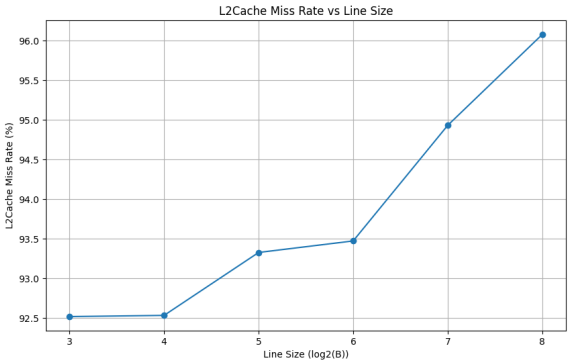

Fig. 8. L2 Cache Miss Rate vs Line size

```
{
    "0.5GHz": "218716824",
    "1GHz": "236600229",
    "2GHz": "267352389",
    "3GHz": "300397331"
}
```

Listing 5. cycles

It is strange. We can find that the the sum of the hits and misses of the caches are exactly the same. But the latencies in ticks are different.

I checked the source code of gem5. All the latency parameters of caches are using cycle as the unit. (Maybe some other components like Bus are using ticks.) Therefore the behavior should have no relation with the clock rate. So the reason of this phenomenon is still unclear.

But the increasing of cycles is not significant. So the execution time is still inversely proportional to the clock rate, generally speaking.

B. Q2

Q: What is the impact of the size of cache block on the miss rate, why?

A: Size of cache block can both increase and decrease the miss rate. According to the results, it will lower the L1I and L1D miss rate. You can tell that the curve is like an upside-down log curve, which indicates that the miss rate has Linear negative correlation with the cache block size, as in Fig.6. The range of size of L1I in the task is far from the diminishing marginal returns of enlarging the cache block size. L1D is similar to L1I, but obvious it has diminishing returns, as in Fig.7.

This is because when the size of the line is increasing, the cache can bear more spatial locality. For example, more jmps in the codes will be included in a single line and will not miss.

When it comes to L2, due to the fixed total size of the cache, the sum of the lines will decrease when you increase the line size, as in Fig.8. L2 handles both the codes and data fetch. So the accesses are more likely to be far. The fewer lines may increase the possibility of the miss.

Notice that the x-axis is logarithm.

C. Q3

Q: What is the impact of the size of cache on the miss rate, why?

A: Increasing the size of cache will always increase the miss rate (Fig.3.4.5.), for given fixed line size, it will simply increase the number of lines, which can reduce the extra miss caused by replacement. Specially, the L2 miss rate has reached a platform when it is 2048kB, as in Fig.5.. This is because replacement will never happen in the tasks when the cache is large enough. So the scaling of cache fails.

V. Acknowledgement

Thanks for the guidance and help from the teacher and the teaching assistant!