Observation of Different Combinations Hardware Architecture with Gem5

1st Hangyul Kim Eric
Department of Computer Science
Virginia Tech
Blacksburg, United States
hangyul@vt.edu

***Abstract*—In the CS5504 Computer Architecture class in the Spring 2024 semester, we should learn that a computer is composed of different combinations of hardware architectures such as CPU, Instruction Set Architecture (ISA), Cache Structure, and Branch Predictor (BP). Since there are many different kinds of these hardware architectures, the performance of a workload can vary depending on the combination of these architectures. In this project, different combinations of CPU, ISA, Cache structure, and BP are tested under a specific workload to determine which combination shows the best performance on that workload using gem5.**

***Index Terms*—gem5, CPU, instruction set architecture, cache, branch predictor**

I. Introduction

To improve performance on a workload, various computer architecture techniques are employed. Using the gem5 simulator, these different sets of hardware architectures can be simulated by passing different parameters through the code.

A. Instruction Set Architecture (ISA)

The two main types of instruction sets are Complex Instruction Set Computer (CISC) and Reduced Instruction Set Computer (RISC). Since these two instruction sets have different structures, they yield different results in terms of performance. In gem5, CISC is implemented with x86, while RISC is implemented with ARM and POWER architectures. In this project, only the x86 ISA is used, as RISC-based ISAs require cross-compilation, which is not functional on our server. We leave a performance comparison between CISC and RISC into future work.

B. CPU

A **pipeline** can be utilized to enhance the throughput of instructions. In gem5, pipelines are implemented using different CPU models: TimingSimple (Simple), Minor, and O3. The Simple CPU, as its name implies, has no pipeline architecture. The Simple CPU can be configured using the following code: `system.cpu = X86TimingSimpleCPU()`. The Minor CPU implements in-order superscalar pipeline. The Minor CPU can be configured with the following code: `system.cpu = X86MinorCPU()`. The O3 CPU has a out-of-order pipeline which requires more resources. The O3 CPU can be used with the following code: `system.cpu = X86O3CPU()`.

C. Cache Structure

A cache structure is a crucial architecture for reducing latency and clock cycles of instructions. With a larger cache size, a computer can increase its cache hit rate, but it also faces an increased miss penalty. We utilize two types of real-world Intel CPU cache structures: Broadwell and Skylake. The detailed specifications of the cache structures are configured in the table I.

D. Branch Predictor

A branch predictor is an important technique to reduce a control hazard. Since the control hazard is a main reason of delay of clock cycles, implementing useful branch predictor requires to improve the performance. In gem5, there are three types of branch predictors: LocalBP, TournamentBP, and TAGE. The LocalBP policy uses n-bit local predictor to resolve control hazard. The LocalBP can be configured using the following code: `system.cpu.branchPred = LocalBP()`. The TournamentBP policy is implemented with both local and global branch history to predict next branch. The TournamentBP can be configured using the following code: `system.cpu.branchPred = TournamentBP()`. The TAGE policy is using tagged geometric histroy length predictor which can predict both short and long branch history pattern. The TAGE can be configured using the following code: `system.cpu.branchPred = TAGE()`.

II. Experimental Work

The experiment is performed on the Virginia Tech rlogin server with SSH configuration. All factors are the same except for CPU, cache structure, and BP. For this project, we selected three workloads familiar to us: Matrix Multiplication (matmul), N-queens (nq), and Lower Triangular Solve (lts). All three workloads are implemented with C code and compiled using the code below.

```
gcc -O0 -ggdb3 -std=c99 -static \
<C_FILE> -o <EXCUTABLE>
```

The `<C_FILE>` field has to be replaced with the name of each workloads' c files (`matmul.c`, `nq.c`, `lts.c`). The `<EXCUTABLE>` field has to be replaced with the name of executable files after compile.

TABLE I
INTEL CPU CACHE STRUCTURES

Cache	Broadwell			Skylake		
	Size (KB)	***Associativity***	***Latencies***	***Size (KB)***	***Associativity***	***Latencies***
L1	32	8	4	32	8	4
L2	256	8	12	1024	16	14
L3	32,768	8	50	16,384	8	50

III. RESULTS

Results are recorded in stats file generated with gem5 command. Each stats files contains many information to measure the performance. The most basic measurement of performance of workload is number of seconds simulated. In stat file, it is shown as `simSeconds` statistics. It shows how many seconds are simulated under certain conditions. Tables below contains actual *simSeconds* of each workloads with several combinations.

TABLE II
SIMULATED SECONDS IN MATRIX MULTIPLICATION

	simSeconds		
Broadwell	***Local***	***Tournament***	***TAGE***
Simple	0.764775	0.764775	0.764775
Minor	0.269586	0.269586	0.269586
O3	0.054199	0.054199	0.053852
Skylake	***Local***	***Tournament***	***TAGE***
Simple	0.765013	0.765013	0.765013
Minor	0.269822	0.269822	0.269822
O3	0.054446	0.054446	0.054092

TABLE III
SIMULATED SECONDS IN N-QUEENS

	simSeconds		
Broadwell	***Local***	***Tournament***	***TAGE***
Simple	0.265261	0.265261	0.265261
Minor	0.099640	0.099640	0.099640
O3	0.029578	0.02797	0.026421
Skylake	***Local***	***Tournament***	***TAGE***
Simple	0.265265	0.265265	0.265265
Minor	0.099644	0.099644	0.099644
O3	0.029579	0.027971	0.026445

TABLE IV
SIMULATED SECONDS IN LOWER TRIANGULAR SOLVE

	simSeconds		
Broadwell	***Local***	***Tournament***	***TAGE***
Simple	0.005220	0.0052205	0.005220
Minor	0.002557	0.002557	0.002557
O3	0.000793	0.000796	0.000796
Skylake	***Local***	***Tournament***	***TAGE***
Simple	0.005227	0.005227	0.005227
Minor	0.002566	0.002566	0.002566
O3	0.000799	0.000798	0.000798

Based on the data above, we should know some vague information about architectures in each workloads. **The O3 CPU consistently outperforms others across all three workloads**. This superiority is expected, given that the O3 CPU utilizes an out-of-order pipeline, which typically results in better performance. An interesting part of the O3 CPU is that it seems like that **only the O3 CPU is influenced by changing a BP policy**. We will deep dive into this in the next section.

For the BP policy, the TAGE policy outperforms in matmul and nq workloads, but **the Local BP policy shows better performance in lts workload**. It will also be analyzed in the next section.

For the Cache Structure, **the Broadwell structure surpasses in all three workloads**.

IV. ANALYSIS

A. O3 CPU Analysis

As mentioned earlier, the O3 CPU consistently outperforms others across all three workloads. This outcome was expected, given the well-known superiority of out-of-order pipelines over in-order pipelines. Unlike other CPUs, the pipeline stages of the O3 CPU is consisted of 5 different stages: Fetch, Decode, Rename, Issue/Execute/Writeback (IEW), and Commit.

With that, the O3 CPU contains more attributes in stat file. Since the O3 CPU consists of the Rename stage in the pipeline stages, it has the *cpu.rename* field in the stat file. The *rename.renamedInsts* shows how many instructions are renamed. In the matmul workload, total 123628226 instructions are renamed. The *rename.runCycles* presents how many cycles are spent during the Rename stage. In the matmul, total 12796562 cycles are spent on the Rename stage. So, we can see that average 9.6 cycles are spent on each renamed instruction. Also, the *rename.serializedStallCycles* represents the number of stalled cycles in the Rename stage. In the matmul, it only has 1009 stalled cycles which is very small.

Also, there are a field *cpu.iew*, which represents IEW stage of the O3 CPU. The *iew.dispatchedInsts* shows the number of instructions that are pushed into the Instruction Queue (IQ) after renamed. In the matmul, there are total 123556609 number of instructions that are pushed which are almost all instructions that are renamed. After they pushed, the *iew.instsToCommit* and *iew.writebackCount* show how many pushed instructions are committed and written-back. In the matmul, the *iew.instsToCommit* and *iew.writebackCount* are 123180682 and 123180275 respectively which shows almost every instructions are written-back.

TABLE V
DRAM STATISTIC WITH DIFFERENT CPUS IN MATMUL

	Simple	***Minor***	***O3***
bwRead (Byte/Sec)	341936	1021772	4951270
rank0.averagePower (mW)	465.57	465.71	466.61

The DRAM statistics can also be the important source of measuring the performance. In the stat file, there is the field *mem_ctr.dram* which shows the condition of the DRAM of the workload. In the Table V, there are two attributes that can shows the performance of the DRAM. The *bwRead* shows the total read bandwidth of the DRAM. Since our workloads mainly perform read operation, we are not considering write bandwidth. As shown, the *bwRead* of the O3 CPU outperforms the others; it is almost 15 times greater than that of the Simple CPU and 5 times greater than that of the Minor CPU. However, for the *rank0.averagePower*, which represents the power consumption, the O3 CPU consumes only 1 mW more power than others.

Therefore, we can conclude that the O3 CPU is superior to others with more complex mechanism and hardware structure but only consumes 1 mW more power.

B. BP Analysis over different CPUs

As we discussed in result section, it seems like only the O3 CPU is influenced by changing the BP policy in terms of the *simSeconds*.

For the Minor CPU, it contains the branch prediction mechanism but it wraps around the branch predictor interface provided by gem5 [1]. Therefore, the performance of the Minor CPU remains unchanged even if we change the BP policy. This has been confirmed by observing the *branchPred* attributes in the Minor CPU stat file. All of the *branchPred* fields in the stat file have a value of 0, which means that branch prediction did not affect the workloads.

For the Simple CPU, unlike the Minor CPU, its *branchPred* attributes have some values in it. Additionally, the values of each *branchPred* attribute differ among the BP policies (local, tournament, tage). As shown in Table VI, the Simple CPU presents meaningful data for each attribute. However, according to the *simSeconds* data in the row for the Simple CPU, it seems that these attributes did not significantly affect its performance. Why? We can find the reason in official document of gem5. For other CPUs, we can assume that the branch prediction is occured in the Fetch stage of the pipeline. However, for the Simple CPU, its fetch stage seems like it doesn't handle any branch prediction mechanism like shown in Fig. 1. Therefore, we should finalize that *branchPred* attributes get some value since it is not wrapped by its own mechanism but it does not actually affect to the cycles of the workloads.

C. Local BP over lts workload

The lts workload shows better performance on the Local BP policy whereas the other two workloads have better results in TAGE BP policy. The reason that the matmul and nq workloads result good performances in TAGE BP policy is simple because TAGE uses several branch predictors that have different in lengths. Since it uses more memory (shown in hostMemory attribute) to predict, it is natural that TAGE superiors others. However, it is hard to tell the reason why Local BP policy shows a better performance in lts workload.

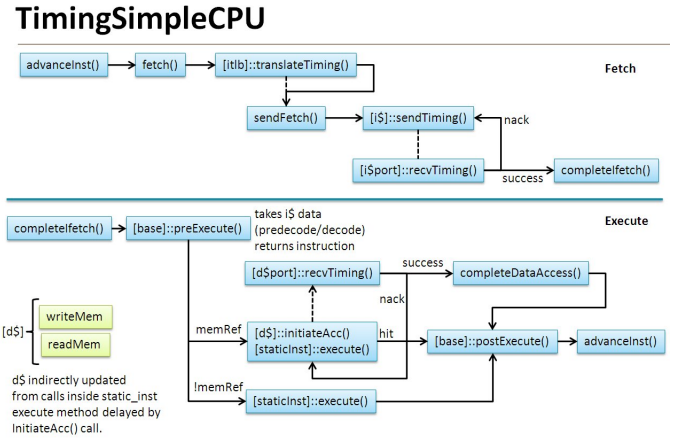

Fig. 1. SimpleCPU figure [2]

The Table VII shows some attributes of the stat files for the lts workload. Similar to other workloads, the *condIncorrect* attribute, which indicates the total number of mispredictions in branch prediction, is larger in the Local BP policy. However, unlike other workloads, the difference is only around 1000, which is relatively small compared to the difference of 16000 seen in the matmul workload. [Table VI] Therefore, we can suppose that the performance of **lts workload has a less dependency on misprediction rate compared to other workloads**.

With that, let's examine another attribute, *fetch.cycles*. The Fetch stage is where branch prediction occurs in the pipeline. As indicated in Table VII, the *fetch.cycles*, which represents the actual cycles of the entire Fetch stage, is lower for the Local BP policy compared to the TAGE BP policy. This is because **the TAGE policy employs a more complex and memory-intensive algorithm, resulting in each prediction taking more cycles compared to the Local BP policy**, which utilizes only a single local branch predictor. Despite the Local BP policy showing relatively good performance in terms of mispredictions, it exhibits only a very slight superiority (0.000003 sec) over the TAGE.

D. Broadwell vs. Skylake

As we shown in result Table, the Broadwell structure consistently outperforms the Skylake structure in all three workloads. The detailed setup for each cache structure is shown in Table I. According to the Table, the L1 cache is same with both structures. For the L2 cache, the Skylake has four times larger size in KB and two times larger associativity than the Broadwell. For the L3 cache, the Broadwell has two times larger size in KB than the Skylake.

In the Table VIII, it shows the *overallMissRate* and *overallMissLatency* of the L2 and L3 caches in the matmul workload. Since the *overallMissRate* and *overallMissLatency* of the L1 cache are same in both structures, they are excluded from the Table. As indicated, the L2 cache miss rate of the Broadwell is slightly larger than that of the Skylake since the Skylake has four times larger size in L2 cache. However, the

TABLE VI
BRANCHPRED ATTRIBUTE IN MATMUL WITH BROADWELL

SimpleCPU	***condPredicted***	***condIncorrect***	***NotTakenMispredicted***	***TakenMispredicted***
Local	2174817	18230	1148	17082
Tournament	2174817	17509	821	16688
TAGE	2174817	1818	1604	214
O3CPU	***condPredicted***	***condIncorrect***	***NotTakenMispredicted***	***TakenMispredicted***
Local	2198144	17970	876	17094
Tournament	2194055	17613	919	16694
TAGE	2178294	1457	1186	271

TABLE VII
ATTRIBUTE IN LTS WITH O3 CPU AND BROADWELL

	Local	***TAGE***
condPredicted	91069	88317
condIncorrect	2424	1304
fetch.cycles	399776	412160

TABLE VIII
CACHE STRUCTURE IN MATMUL WITH SIMPLECPU AND LOCAL BP

	Broadwell	***Skylake***
l2cache.overallMissRate	0.035636	0.035627
l2cache.overallMissLatency	766984000	782900000
l3cache.overallMissRate	0.999755	1
l3cache.overallMissLatency	660671000	660320000

difference is very small (0.000009) so it can be considered negligible. In contrast, the difference between *overallMissLatency* of the L2 cache is huge; the Broadwell has much lower miss latency than the Skylake. **By increasing the size of cache, the miss rate can be decreased but the miss penalty is increased by trade-off**. Although the Skylake has smaller size of the L3 cache (also, smaller *overallMissLatency*), it won't affect to the performance that much since the hit rate of the L3 cache is almost (or exactly) zero. Therefore, we should conclude that the Broadwell has better cache structure policy than the Skylake for these workloads.

CONCLUSION

By analyzing several combination of the hardware architectures (CPU, Branch Precdiction policy, and Cache structure) with three workloads (Matrix Multiplication, N-queens, and Lower Triangular Solve), we should know how the gem5 simulator measure the performance and what factors affect to the workloads performances.

For the CPU, the O3 CPU consistently outperforms the other two CPU, the Simple CPU and the Minor CPU since it has more complex and effective mechanism on the pipeline stages.

For the Branch Prediciton policy, the TAGE policy is superior on the matmul and nq workloads whereas the lts workload has a better performance with the Local BP policy. Since the TAGE policy has more complex and memory-intensive algorithm, it shows better performance in the matmul and nq workloads, which require accurate branch prediction, but the lts workload, which does not have many branch prediction, has a better performance with the Local BP policy that is more simple.

For the cache structure, the Broadwell consistently shows better performance over the Skylake. Since the miss penalty of the Skylake is larger than the hit rate of it, the trade-off for the Skylake is worse than that of the Broadwell.

REFERENCES

[1] "Minor CPU model," gem5, https://www.gem5.org/documentation/ general_docs/cpu_models/minor_cpu (accessed May 2, 2024).

[2] "SimpleCPU," gem5, https://www.gem5.org/documentation/general _docs/cpu_models/SimpleCPUbasesimplecpu (accessed May 2, 2024).

[3] "O3CPU," gem5, https://www.gem5.org/documentation/general _docs/cpu_models/O3CPU (accessed May 4, 2024).