Testing system performance with and without cache memory using GEM5 simulator

Kashi Vishwanath Kolloju
dept. of Electrical and Computer Engineering
Virginia Tech
Blacksburg, Virginia
kkashivishwanath@vt.edu

***Abstract*— In this project, performance of a system is being evaluated using cache memory by simulating on GEM5. GEM5 simulator has been extensively used in computer architecture simulation and in evaluation of architectures for high performance computing (HPC). We simulate gem5 on x86 architecture (primary ISA used in most HPC systems). The Out-order CPU (O3) with Ruby memory model is used in a Syscall emulation X86 environment with Linux OS. First, we design a basic system without cache and evaluate the performance of that system by running micro benchmarks. Secondly, we add instruction and data cache with varying sizes to the system and run benchmarks. Finally, we evaluate the Performance difference observed by both systems and prove that caches are indeed useful in multi-architectures and for system throughput.**

Keywords—gem5, spec, x86 ISA, cache size, instruction cache, data cache, micro-benchmark

I. INTRODUCTION

Instruction and data caches are well known architectural developments that allow significant improvement on the performance of high-end processors. Earlier CPUs with Cache had only one level of cache later, level 1 caches split into L1D for data and L1I for instructions. Cache Memory affects the execution time of a program. Cache memory [11] gives data at a very fast rate for execution by acting as an interface between faster processor unit on one side and the slower memory unit on the other side. In Cache memory frequently used data or instructions are kept so that it can be accessed at a very fast rate improving overall performance of the computer. There are various level of cache existing in a computer system, the cache at last level is being largest and slowest, cache at first level is fastest and smallest. Commonly in a processor first level cache (L1) reside in the processor and second level cache (L2) and third level cache (L3) are on separate chip. In multi-core processors, each processor has its own L1 cache. Whenever CPU wants any data or instruction it sends address to the cache memory, address is searched in cache memory firstly, if data is found in cache it is given to the CPU. Finding a data in cache memory is called as cache HIT. If data is not found in cache then it is called as cache MISS, in this case address is searched in main memory for the data or instruction. After data is received from main memory, a block of data is transferred from main memory to cache memory so that all further request can be fulfilled from cache. Performance of cache memory is measured in HIT Ratio. Hit Ratio is No. of Hits / No. of Hits + No. of Miss. The hit ratio approaching towards 1 is considered as best. gem5 supports several ISAs (instruction set architectures) like x86, ARM, Alpha, MIPS, RISC-V and SPARC. It supports simple and quick functional simulation and detailed simulation modes. The detailed simulation can be either with system call emulation or with full-system support, in which gem5 is capable of booting a Linux operating system. gem5 supports two different pipeline implementations, in-order (IO) and out-of-order (OOO). Built and simulate on personal computer with System call emulation mode instead of Full system mode and Ruby memory model.

II. RELATED WORK

The memory wall [1] refers to the increasing gap between the speed of logic and the speed of main memory, first spelt out by Wulf and McKee in 1995 [WM95]. However, this gap has existed for a long time. With the introduction of semiconductor memories in the early 1970s, memory performance caught up substantially but still lagged logic performance. Since then, the focus on developing memory technology has been on increasing density rather than decreasing latency (indeed the two are somewhat contradictory), and the gap between main memory speed and processor speed has been ever widening. In addition, recent trends and microarchitectural innovations such as multithreading and multiple cores have only increased the pressure on the memory subsystem.

In the paper [2] an accurate analysis of the effects of soft errors in the instruction and data caches of a soft core implementing the SPARC architecture.

A cache organization [3] essentially eliminates this penalty. This cache organizational feature has been incorporated in a cache interface subsystem design, and the design has been implemented and prototyped. A master-slave cache system has a large, set-associative master cache, and two smaller direct-mapped slave caches, a slave instruction cache for supplying instructions to an instruction pipeline of a processor, and a slave data cache for supplying data operands to an execution pipeline of the processor. The master cache and the slave caches are tightly coupled to each other.

This tight coupling [4] allows the master cache to perform most cache management operations for the slave caches, freeing the slave caches to supply a high bandwidth of instructions and operands to the processor's pipelines.

An automatic tool-based approach [5], to bound worst-case data cache performance. The given approach works on fully optimized code, performs the analysis over the entire control flow of a program, detects and exploits both spatial and temporal locality within data references, produces results typically within a few seconds, and estimates, on average, 30% tighter WCET bounds than can be predicted without analyzing data cache behavior.

According to the study [6], a cache memory may contain contents that are susceptible to corruption. A cache controller, with the use of a threshold timer, may employ various operations to flush modified cache contents into a main memory and invalidate cache contents so that they are overwritten. Some operations include periodically flushing and invalidating the whole cache memory, periodically flushing and invalidating modified contents, and periodically flushing and invalidating contents based on the time saved in the cache memory. By overwriting cache contents that might otherwise be constantly stored in the cache memory, the system minimizes the probability of cache contents becoming corrupt. The periodic updating of the main memory may also increase the probability of successfully recovering from potential cache parity errors while still maintaining high performance associated with using a cache memory.

III. METHODOLOGY

We will be using MESI Two level Protocol. This protocol models two-level cache hierarchy. The L1 cache is private to a core, while the L2 cache is shared among the cores. L1 Cache is split into Instruction and Data cache. The protocol has four types of controllers L1 cache controller, L2 cache controller, Directory controller and DMA controller. L1 cache controller is responsible for managing L1 Instruction and L1 Data Cache. Number of instantiations of L1 cache controller is equal to the number of cores in the simulated system. L2 cache controller is responsible for managing the shared L2 cache and for maintaining coherence of on-chip data through directory coherence scheme. The Directory controller act as interface to the Memory Controller/Off-chip main memory and responsible for coherence across multiple chips/and external coherence request from DMA controller. DMA controller is responsible for satisfying coherent DMA requests[1].

CPU Parameters: We are going to use O3 as our pipelined, out-of-order CPU model for the simulation.

Memory Model: We will be using Ruby option in the highly configurable se.py script by Gem5, due to its fidelity and Cache Coherence flexibility. Ruby accurately models both cache coherence and network related features in the memory system. We experiment with L1I, L1D, L2 cache sizes and evaluate the performance of the system.

Range for L1 Instruction cache: L1I cache ranges from 16 KB to 1,024 KB. L1I - Default: 64 KB & Experimental range: [16, 64, 256, 512, 1024] kilobytes.

Range for L1 Data cache: L1D cache ranges from 16 KB to 1,024 KB. L1D - Default: 64 KB & Experimental range: [16, 64, 256, 512, 1024] kilobytes.

We are going to write a program using the parameters we discussed above. Now, we run gem5 using our program on microbenchmarks. We will find the detailed simulation report of the run at stats.txt. Using this file, we will determine the performance parameters of the system.

IV. EXPERIMENTAL RESULTS

We extract values from stats.txt to calculate IPC values by running micro benchmark(lfsr, mm, Mat-mul, merge)[9][12]. We wrote a .py code which takes arguments as locations of benchmarks. We calculate IPC by noting down number of instructions run and cycles elapsed for each benchmark from stats.txt. We run all the benchmarks first on simulated system having no cache capability. Figure1 compares IPC values of system without any caches.

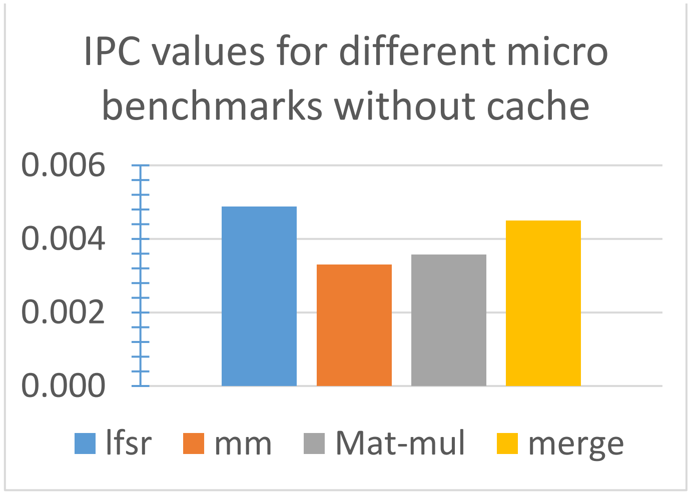

Figure 1

Now, we run all the benchmarks on simulated system having cache capability. We defined this system with L1instruction cache size of 16kB and L1data cache size of 16kB and L2 cache size of 64kB. We also set cache associativity of L1 cache as 2 and L2 cache as 8. Figure2 compares IPC values of system with caches for different benchmarks

[1] http://www.m5sim.org/MESI_Two_Level

We can observe that adding a cache to a model increased the IPC values. As we can see for ‘lfsr’ benchmark IPC increased from 0.00487 to 0.08269 , for ‘mm’ benchmark IPC increased from 0.00329 to 0.23 instr/clock and for ‘Mat-mul IPC increased from 0.0035 to 0.26 instr/clock and for ‘merge’ benchmark IPC increased from 0.0044 to 0.31 instr/clock. From this analysis it is obvious that system with cache has improved its performance on the ‘lfsr’ benchmark. In Stats.txt file we can notice various types of parameters using those we can find out the miss rates, hit rates etc., In the README file attached it has information on how to build and run benchmarks using gem5.

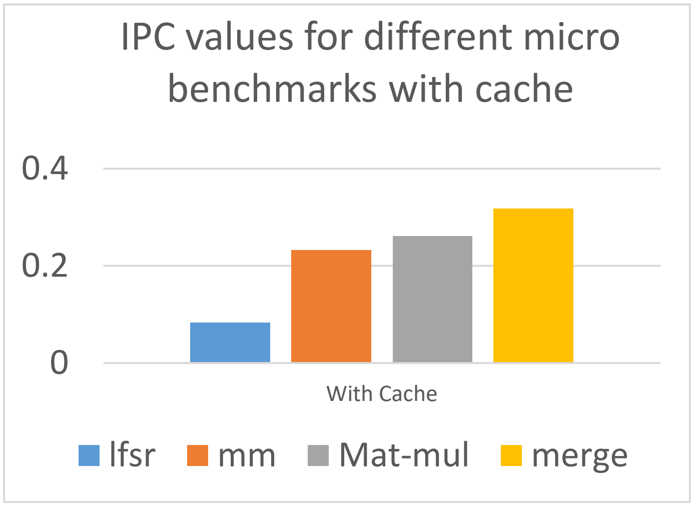

Figure 2

V. DISCUSSION

In this paper we have implemented by experimenting on Improvement of systems with cache have been discussed by running benchmarks on a gem5 simulated system on Linux platform. System model built in X86 (because it is primary architecture used in high performance computing). In future work, we can design models varying cache sizes and associativity in different architectures. We can also implement gem5 for finding sources of inaccuracies in micro-benchmarks

VI. REFERENCES

[1] https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-800.pdf

[2] Rebaudengo, Maurizio, M. Sonza Reorda, and Massimo Violante. "An accurate analysis of the effects of soft errors in the instruction and data caches of a pipelined microprocessor."Proceedings of the conference on Design, Automation and Test in Europe-Volume 1. IEEE Computer Society, 2003.

[3] Kroft, David. "Lockup-free instruction fetch/prefetch cache organization." Proceedings of the 8th annual symposium on Computer Architecture. IEEE Computer Society Press, 1981. https://dl.acm.org/citation.cfm?id= 801868

[4] Cohen, Earl T., et al. "Master-slave cache system for instruction and data cache memories." U.S. Patent No. 5,551,001. 27 Aug. 1996. https://patents.google.com/patent/US5551001A/en

[5] White, Randall T., et al. "Timing analysis for data caches and set-associative caches." Proceedings Third IEEE Real-Time Technology and Applications Symposium. IEEE, 1997. https://ieeexplore.ieee.org/document/601358

[6] Koktan, Toby. "Cpu instruction and data cache corruption prevention system." U.S. Patent Application No. 12/574,612. https://patents.google.com/patent/US20110082983A1/en

[7] Binkert, Nathan; Sardashti, Somayeh; Sen, Rathijit; Sewell, Korey; Shoaib, Muhammad; Vaish, Nilay; Hill, Mark D.; Wood, David A.; Beckmann, Bradford; Black, Gabriel; Reinhardt, Steven K. (2011-08-31). "The gem5 simulator". ACM SIGARCH Computer Architecture News. **39** (2): 1. doi:10.1145/2024716.2024718.

[8] http://learning.gem5.org/book/presentation_notes/part1.html

[9] https://github.com/bhagyashreeborate/Gem5-Simulator

[10] https://arxiv.org/abs/1911.11642v1

[11] Vivek Chaplot.” Cache Memory: An Analysis on Performance Issues”. International Journal of Advance Research in Computer Science and Management Studies Volume 4, Issue7, July2016. http://www.ijarcsms.com/docs/paper/volume4/issue7/V4I7-0014.pdf

[12] https://github.com/PolyArch/cs251a-microbench