An Empirical Study for Multi-Level Cache Associativity

Dr. Hassan Rajaei, Bowling Green State University

Hassan Rajaei is a professor of computer science at Bowling Green State University, Ohio. His research interests include cloud computing, High Performance Computing (HPC), distributed simulation, parallel and distributed processing, communication networks, wireless communications, and IoT. Rajaei received his Ph.D. from Royal Institute of Technology, KTH, Stockholm, Sweden, and he holds a M.S.E.E. from the University of Utah, and a BS from University of Tehran.

An Empirical Study of Multi-Level Cache Associativity

Abstract

Most CPUs architecture use multi-level caches with different associativity. A cache plays an essential role by providing fast access to the instructions and data to improve the overall performance of the system. To demonstrate the complexity of the issue in an advanced computer architecture course, we used an empirical simulation study to focus on performance of multi-level caches and their associativity. This paper presents the result of such study. Designing and predicting caches behavior has been subject of numerous simulation studies. Cache simulation tools provide support for diverse configurations of the system with multiple scenarios to ensure the system could perform at peak level. In this study, we explore multi-level caches and their performance when changing various associativity and block size.

We surveyed several simulation studies specialized for cache design and processor performance. We chose two tools, CACTI 5.3 and SimpleScalar 3.0, to empirically study and analyze numerous configurations. We cross-examined the base model by adding cache associativity to check performance implications of each configuration. CACTI helped to compare several scenarios based on access time, cycle time, and other factors. SimpleScalar helped to better analyze the results based on factors such as total number of instructions executed and hit- or miss-rates. In addition, we combined the access time from CACTI the one from SimpleScalar to get total number of Access per second. Our results were scrutinized against others and found that in most cases they were similar or slightly improved for multilevel cache associativity.

1. Introduction

Caches are high-speed buffers acting as local memories for CPU to store recently or most frequently accessed instructions and data fetched from RAM memory. A chase reduces the average access time of the main memory as well fetch latencies. In recent CPUs caches have multi-levels and associativity. The concept of multi-level cache is illustrated in Figure 1. The time taken to access the main memory can range from 300 to 600 nanoseconds while accessing the cache only takes 50 to 100 nanoseconds [1]. This is significant reduction where numerous designers and researchers are aiming even higher performers.

The CPU generates a virtual address that is passed onto the Translation Look-Aside Buffer (TLB) and the cache memory. The TLB often use hashed number to determine whether the target line is in a cache. If a match is not found in any cache, the main memory is access, which further delays the needed info for CPU.

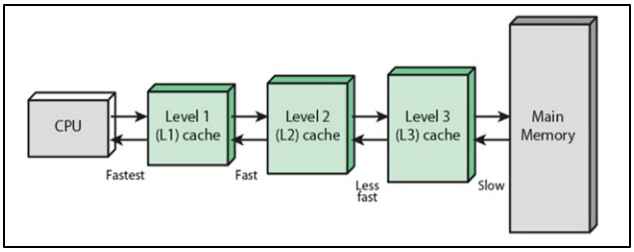

Figure 1: Overview of a Multi-level cache system

Cache coherency grows with multilevel caches and cores, thus we need careful performance studies. Numerous studies aimed at reducing the complexity of cache coherent hardware. Our work focuses on understanding the complexing and study performance of multi-level cache, without using any specific system. Cache simulation tools provide support for diverse configurations of the system and help to capture the real world scenarios to ensure that the system performs at an optimal level.

We surveyed cache simulation studies to better understand the needs for cache simulation. Then, we designed numerous scenarios using different cache configuration and sizes to reflect the scalability. Keeping the focus on achieving maximum performance, cache associativity is also observed and extensively studied to verify the gains in performance were made possible. Various types of cache associativity were examined and their benefits and limitations are summarized. We also studies that the relationship between cache associativity and cache coherency. One result [2] suggests that higher associativity can reduce miss rate. Another result [3] indicates that miss rate from lazy write impacts the cache coherence problem. Further, some results show that the miss rate in 8-way set associativity is almost same in the fully associative, and the fully associative cache has greater delay which opposes the high speed calculation design principle.

We chose CACTI 5.3 [4] and SimpleScalar 3.0 [5] for the cache configuration performance studies. CACTI helped to compare various configurations based on access time, cycle time, area on chip and area efficiency. SimpleScalar helped to understand the comparisons basis of the total number of instructions executed, rate at which instructions are executed, total number of hits, total number of misses, the miss rate, total number of pages allocated per program, size of the pages allocated and total page table accesses. In addition, we combine the results of CACTI 5.3 and SimpleScalar 3.0 to get better picture of the system. The Access Time obtained from the simulation in CACTI 5.3 and the total numbers of Access obtained from the simulations in SimpleScalar 3.0 are combined to give the total number of Access per second.

The rest of the paper is organized as follow: In Section 2 related work is reported. Section 3 overviews Cache Associativity, whereas Section 4 gives some details on Multilevel Caches. Section 5 shows taxonomy of cache simulators and Section 6 gives the details of Simulation Results. Section 7 gives concluding remarks.

2. Related Work

Hill and Smith [1] investigated the effects of associativity of caches. They created a new and efficient algorithm called Forest. They used this algorithm to have two investigations. One was to

simulate alternative direct mapped cache, n-way set associative cache and fully associative cache. The second was to analyze the effect of different cache associativity on cache miss rates by using their new algorithm. After that, they introduced all-associative methodology for simulating alternate direct-mapped, set-associative, and fully-associative caches based on the 'Stack' algorithm. The space complexity of the all-associativity simulation is O(Nunique). In their experiment, they showed that higher associativity can reduce miss rate.

Duska et al [3] investigated the impact of cache coherence on hit rate. They analyzed the relationship between cache size and hit rate. The range of second-level cache hit rates vary from 24% to 45% and most of the second-level cache hit rates level off at cache sizes smaller than 10 GBs. Further, they analyzed the relationship between request rate and hit rate. They examined how increasing request rate will increase hit rate. Thirdly, they analyzed the relationship between hit rate and cache coherence. Fourthly, they analyzed Web-client sharing, sharing between clients from different traces.

Przybylski et al [6] analyzed the benefits of utilizing multi-level cache memory systems in modern day computers. They found the primary level of the cache helps in reducing the number of references made to the second level, which helps in increasing the overall performance due to the reduced number of misses at the primary level. It was observed that the secondary level of the cache does not necessarily have to be as fast as the primary, rather it should have a bigger size to store more data for easier memory access. Conte et al [7] proposed a single pass method for modelling multi-level caches which is faster than the traditional techniques. By increasing the number of levels in cache, the performance also increases. Powerful multi-core processors as the task can be divided into smaller parts for faster processing. They used tracing techniques to reduce the cache miss rate which increased the overall performance.

Rabin et al [8] simulated set associative caches by proposing a single pass technique which works on caches having similar line size but different associativity and number of sets. By utilizing a binomial tree, an efficient searching algorithm was developed which reduces the number of comparisons, thereby, decreasing the computation time. Performance gains in the factors of 1.2 to 3.8 were observed over the traditional simulation techniques. Hardy et al [9] analyzed cache performance by using a set of static instructions which was implemented with small and large program sizes. By calculating the worst case execution times, they were able to clearly observe the benefits of utilizing a multi-level cache over single level.

With the increasing number of applications and programs that are executed on the computer system with varying levels of complexity, scalability of cache memory was necessary to keep the performance optimal and efficient. By making use of MorphCache [10], the researchers have been able to make the cache memory system act as a virtual processor to supplement the core processor while executing the different sets of instructions. It was observed that average throughput was increased by 29.9% for shared cache levels and 27.9% for private cache levels. Kecheng et al [11] discussed cache performance, specifically of out-of-order processors. They examine non-blocking cache, which merges different cache misses with the same cache-line address and overlaps multiple outstanding memory requests into one memory request to reduce the average cache miss penalty. They address the problem of time-consuming simulations to quantify memory-level parallelism. They propose a cache performance evaluation to fast estimate MLP using analytical

models and trace analysis. They perform simulations using Gem5 simulator. They compare their results with previous benchmarks and attain desired results.

Yavits et al [12] proposed a cache optimization model which analytically varies the access time of a cache as a function of its size. In their model, cache sizes vary from 64 Kb to 4 Mb. They also employ CACTI cache simulator and conclude that it serves as an efficient tool for the design architect for cache area and access time estimations, including line size, number of banks and aspect ratio optimization. Waldspurger et al [13] proposed a multi-model cache simulation model which performed scaled-down simulations of complex multi-workload environments of a cache. They advocate the effectiveness of Miss Ratio Curves (MRCs) which can be updated in real time. MRCs helped them to model non-LRU algorithms, providing them with extremely accurate results while requiring less time and space then a full simulation. They also introduced a new approach called SLIDE which improved miss ratios by employing the concept of scaled- down MRCs.

Hachem et al [14] explored the concept of a coded cache structure and multi-cast broadcasting through a single cache. Coded caching helps to optimize storage costs of the data being stored in Access Points of a network and transmission cost of the data being transmitted by Broadcast points. They help achieve a trade-off between the two based on two basic models, namely, multi-user popularity and access model and single-user popularity and access model. The multi-user model allows multiple users accessing the cache memory at a single access point, whereas, single-user model only allows access to a single user at a given moment in time. The caches at these access points are single level, they have uniform data distribution. The data is prioritized on the basis of popularity. They also explored the concept of multi-level caches at these access points and derive information-theoretic outer bounds in order to evaluate the performance of the proposed schemes. Maeda et al [15] addressed the single locality granularity problem of using reuse distance as a parameter for analytical miss rate estimations and synthetic trace generation in multi-level caches. They proposed a generalization of the reuse distance, namely, Hierarchical Reuse Distance (HRD) which solves the above mentioned problem. Their method had the same synthetic complexity and profiling as the traditional reuse distance method. However, they observed that average miss rate error in sectored caches was reduced by a factor of 3 when they employed HRD. They also observed that their method enhanced the performance of L2 and L3 cache levels by a factor of 4.

Zhang et al [16] worked on the analysis of multi-level write-back caches for Worst Case Execution Time (WCET). Their main focus was firstly determining whether a block of data in a shared cache was dirty, i.e. not up to date, and then estimate a safe write-back window which updates that block of data in the cache. They treat each write-back as a cache access to the lower level of the cache. They also perform simulations to empirically analyze their approach against the work of previous researchers. Shatnawi et al [17] proposed a new cache design called "set folding", which alleviates the need of using a higher associative cache to achieve the benefits of high hit rate, lower conflict misses and lower access times. They proposed to divide the cache in two subsets, namely, the exclusive subset which hosts data blocks that have the matching indices with the set index and the shared subset which hosts data blocks with the matching indices as well as different indices. They also empirically evaluated their design by simulating it using SESC microprocessor architectural simulator on SPEC2006 benchmarks.

3. Cache Associativity Overview

Cache associativity reserves multiple cache sectors for each potential address set. Based on different forms and capacities of a cache, it can be divided in three basic types: direct mapped, fully associative and n-way set associative. In a system, when many different devices share a common memory resource, if the data in the cache is incoherent, it will cause problems. This problem is particularly noticeable in multi-processor systems. Cache coherency preserves shared data in multiple caches and maintains data uniformity [2].

3.1. Direct Mapped Cache Associativity

This is a many-to-one mapping relationship in which each data block in the main memory has only one place in the cache corresponding to it. Direct mapping is also called one-way associative. The cache maps each address of the main memory to a single location of the cache. Multiple addresses of main memory may be mapped to the same location in the cache. Thus, this leads to an increase in the miss rate of cache, consequently has lowest hit rate of all three cache associativity. However, its implementation is the simplest, and matching the fastest speed.

3.2. Fully Associative Cache

In this architecture, any address in the memory can be stored anywhere in the cache. This cache is very flexible, but usually hard to achieve. The fully associative cache is sharply compared with the direct mapping and n-way set associative cache. The Full associative has highest hit rate, but the CPU would have to compare the memory address with a large number of cache tags in order to access cache which is main memory mapped. Thus, this degrades the efficiency of the cache, and the internal design is very complex.

3.3. N-Way Set Associative Cache

The n-way set associative is a combination of the two approaches described above. The main memory and cache is divided into n set, each set has the same number of blocks. There is a direct mapping method between the set and the block, and the two sets are internally connected using fully mapping method. Multiple researchers found that 8 is the magical number in the n-way set associative because 8-way set associative has almost same hit rate as it is in a fully associative cache. If associativity is more than 8-way, the delay will increase while conflicts possibilities will increase.

3.4. Miss Rate/Hit Rate

Cache miss is a failed attempt to read or write a piece of data in the cache. Cache miss causes the main memory access to have a longer delay. There are three cache misses: instruction read errors, data read errors, and data write errors. Different cache associativity affects the miss rates in the cache. The relationship between miss rate and cache associativity we have already discussed above. Higher associativity can reduce the miss rate and, but in turn, higher miss rate will cause cache coherency problem.

4. Multi-Level Cache

A key advantage of using multi-level cache is to increase memory bandwidth of the processor by providing different levels for data processing. Furthermore, the average memory access time is also improved as the processor can decide which level of the cache to use for data processing depending on the priority and the number of instructions of the task. Multi-level caches are the main components in most computers based on good performance while minimizing the associated costs. The following subsections discuss various levels of the cache memory and also properties which have an effect on the cache performance.

4.1. Level 1 Cache

The cache closest to CPU is called Level 1 (L1). This cache is located on the chip and data can be organized in 64 byte blocks. It is the fastest form of temporary storage after registers. The delay interface is zero wait-state to the processor's execution unit because it is located on the chip but it is limited in size. The size of L1 cache is kept small to ensure faster access to the data due to the small amount of area that needs to be searched by the processor. This can lead to an increased miss rate due to small amount of storage space available which can be dealt by the secondary cache levels.

4.2. Level 2 Cache

Today, most computers add a Level 2 (L2) as a secondary cache. The L2 cache is generally larger than the L1 with large blocks to store more data. The focus is kept on reducing the number of misses rather than access time to compensate for the misses that occurred in the L1 cache. L2 cache tends to be slower than L1 cache due to the larger size and also cost constraints. It acts as a buffer to pre-fetch data from the main memory making it available to the processor ahead of time to avoid any delays. L2 caches were introduced in Intel Pentium processors and have been widely used since then. The usual size of L2 is about 256KB or 512KB.

4.3. Level 3 Cache

While L1 & L2 are often inside a single core, most multicore system add a third level, L3 cache that is shared with all cores using an internal bus or ring network. Cache coherency is mostly maintained by L3 cache.

4.4. Cache Size and Latency

The size of the cache has a major impact on the overall system’s performance. As cache memory is faster than the main memory, having a larger cache will lead to more storage space for faster accessible data. But implementing a really large cache is a complex process as there are various constraints like space and cost. By increasing the size of the cache, the various levels need to be made up of fast SRAM to reduce data latency which is expensive. The problem of cache coherency will also come into play, thereby, further increasing the complexity of implementing such a hardware system. It also depend on the type of program or application being executed, whether it will be able to take advantage of a large cache or not. While the idea of increasing the size of the

cache does sound enticing for improving performance, the limitations and complexity overshadow the supposedly improved performance gains.

5. Cache Simulators

Cache Simulations can be achieved with help of various detailed architectural simulators often available as open-source tool. They can support simulating variety of different system configurations without the need of actually investing time and money to build the real system. The results obtained by conducting these simulations can be used to analyze the system at both a macro and microscopic level to ensure optimal efficiency. Cache simulations can be classified into several categories each having its own sub category as shown in Figure 2. The classification uses numbers of properties of the tools such as execution parameters, supported architecture, and simulation time. The classification leads to easier analysis of various tools and choosing the right tool for the situation.

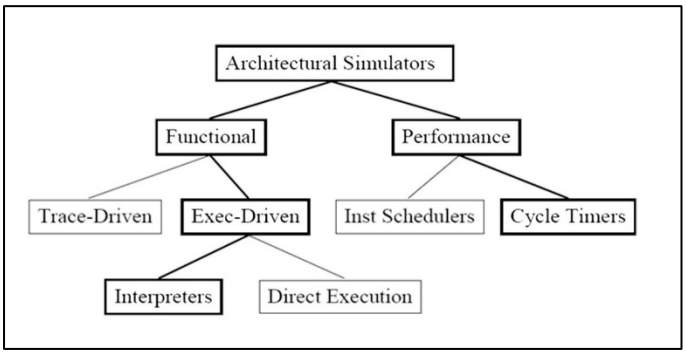

Figure 2: Taxonomy of Cache Simulation Tools [5]

6. Experimental Setup and Results

We present a cache model to estimate the access latency, cycle time, total area on chip and area efficiency with the help of CACTI 5.3 simulator. Further, we compare the total number of instructions executed, rate at which instructions are executed, total number of hits, total number of misses, the miss rate, total number of pages allocated per program, size of the pages allocated and total page table accesses using SimpleScalar 3.0 simulator. Moreover, we compare the Total number of Accesses per second by each cache configuration. We take the total number of accesses obtained using SimpleScalar and divide it by the access times obtained using CACTI.

6.1. CACTI 5.3

Using CACTI 5.3 simulator we compared cache configurations on the basis of cache associativities and number of banks. We chose direct mapped, 2-way associativity, 4-way associativity and 8-way associativity to conduct our simulations. Similarly, we varied the number of banks per simulation: 1, 2, 4, 8 for each associativity. In our simulations we have kept, the total cache size as 256KB, block size as 64 bytes, only 1 read/write port and chip technology as 60 nm, constant. We also compare our simulation results with the results of Gundai et al [18]. They chose a 64KB cache size for direct mapped, 128KB cache size for 2-way associative cache, 256KB cache size

for 4-way associative cache and 512KB cache size for 8-way associative cache. They use only one read/write port with 90nm technology and only 1 bank per cache. The cache shown in Figure 3 is based on [18]; a 4 bit-sliced cache. However, input and output latches are not shown in this figure. Bit-sliced caches have greater access times. They make use of pipelining to hide this delay.

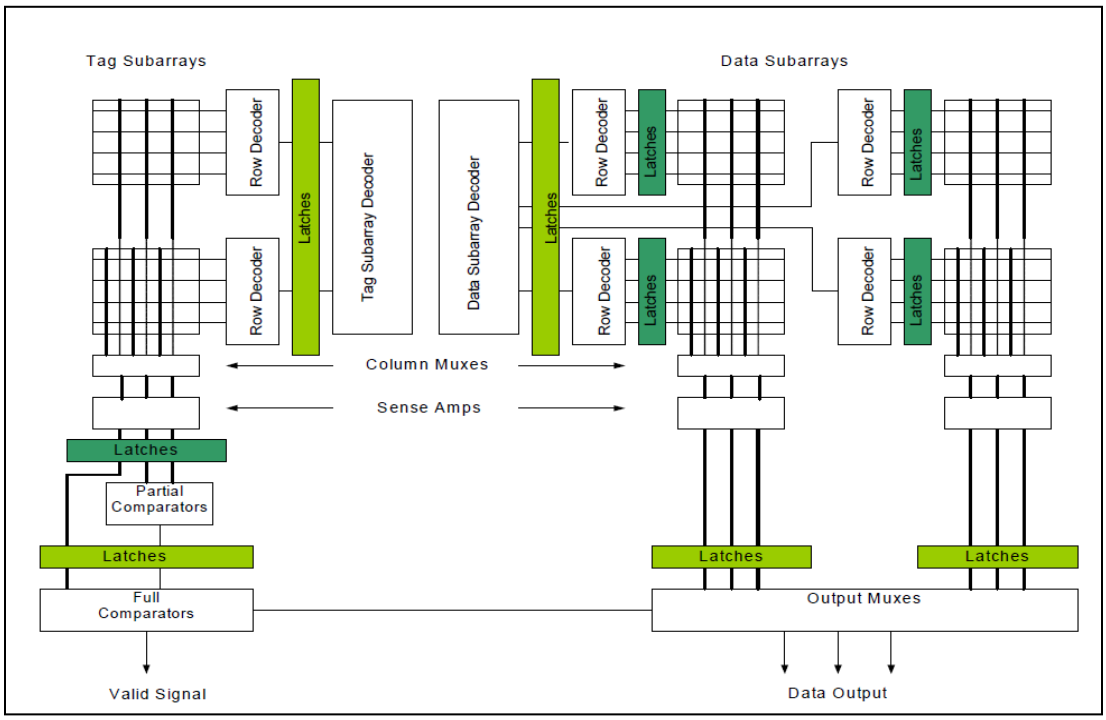

Figure 3: Cache structure used in [18], were our study compares with them.

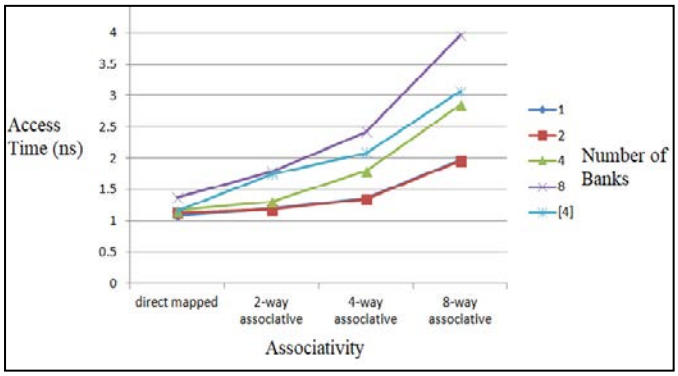

Figure 4: Access Time (ns) per cache configuration

Figure 4 shows the variation in access times, in nanoseconds, of various cache configurations when we take different associativity with variable number of banks. As the number of banks increase, the access time also increases exponentially and direct mapped associative caches have the least access time as compared to any other associativity. This is due to the fact that with multi-associative cache the entry you are looking for can be stored in multiple locations. Therefore, to

search for an index entry you need to search at multiple locations inside a cache. Moreover, more cache banks mean even further division of the cache. This results in more searching of addresses inside a cache which further increases access time. The simulation performed by [18] used only one bank per cache. A rough comparison of the access time obtained in their experiment and has approximately 33% less access time.

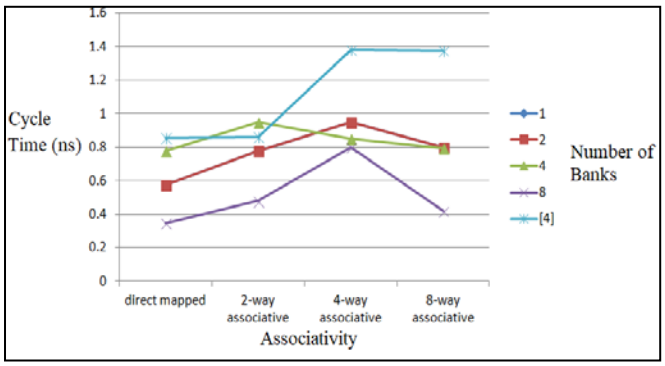

Figure 5: Cycle Time (ns) per cache configuration

Figure 5 shows variation in cycle times, in nanoseconds, of various cache configurations when we take different associativity with variable number of banks. As the number of banks increase, the cycle time decreases and direct mapped associative caches have the almost the same cycle time as compared to 8-way associative cache, while cycle times of 2-way and 4-way associative caches are more than the former two associativity. More number of banks means there can be a scope for more parallelization inside the cache, therefore, the cycle time decreases with the increase in the number of banks. For direct mapped associative caches the cycle time is less than 2-way and 4-way associative caches because there is only one location in the cache where an address index refers to, meaning there is no subdivision. While in 8-way associative cache, the cycle time is similar to a direct mapped cache because the cache is converted virtually into many direct mapped caches which decreases the cycle time. This is possible due to the tradeoff between associativity and area required for the cache on the chip. The cycle time of the configuration used by [18] is 50% higher than our configuration when using one bank per cache.

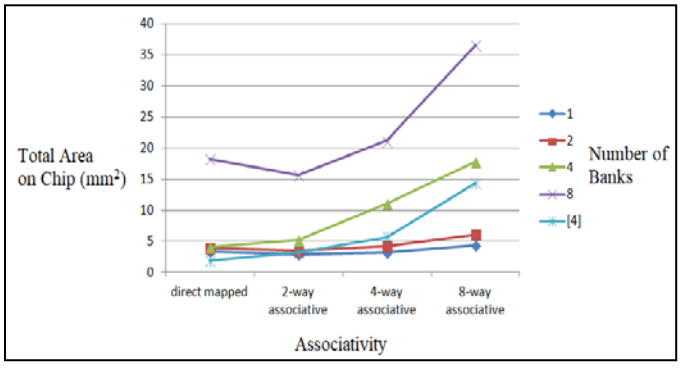

Figure 6: Total Area on chip (mm^2) per cache configuration

Figure 6 shows variation in Total Area on Chip, in mm2, of various cache configurations when we take different associativity with variable number of banks. As the number of banks increase, the total area required on the chip for the cache also increases exponentially and 2-way associative caches require the least area on chip as compared to any other associativity in most cases. Area on chip increases with associativity because apart from index address, there is now a tag address also for each enter, which increases the area needed to store the data. When compared to the cache configuration used by [18] and taking into consideration only 1 bank per cache, our configuration's area on chip is almost the same for 2-way and 4-way associative caches, while, direct mapped cache needs slightly lesser area on chip with their configuration. Also, for 8-way associative cache, our configuration is saves about 60% area on chip.

6.2. SimpleScalar 3.0

Using a custom configured version of SimpleScalar 3.0, we compare the total number of instructions executed, rate at which instructions are executed, total number of hits, total number of misses, the miss rate, total number of pages allocated per program, size of the pages allocated and total page table accesses. We chose a cache configuration with L1 instruction cache size as 64KB, L1 data cache size as 256KB and L2 unified cache size as 1MB. We have kept only 1 read/write port, 1 bank and chip technology as 60 nm, constant. We analyzed cache performance by varying associativity, i.e., direct mapped, 2-way associative, 4-way associative and 8-way associative and the levels of cache, i.e., only L1 instruction cache, both L1instruction cache and L1 data cache and a combination of L1instruction cache, L1 data cache and L2 unified cache.

The obtained results show that a total of 33 pages were allocated for the program we simulated, a total of 1545833 page table accesses were executed and total size allocated to the pages, also called table size, was 132 KB. We also compare our results with the results of [19] and [20]. In [19] the authors chose 4-way associative L1instruction cache and L1 data cache, both 32Kb in size and a block size of 64 bytes, and an 8-way associative L2 unified cache of size 512KB. They generate a page table of size 4KB. In [20], the authors chose a direct mapped 16KB L1 instruction cache, a direct mapped 32KB L1 data cache and a 4-way associative 256 KB L2 unified cache. All the cache levels had a block size of 64 bytes.

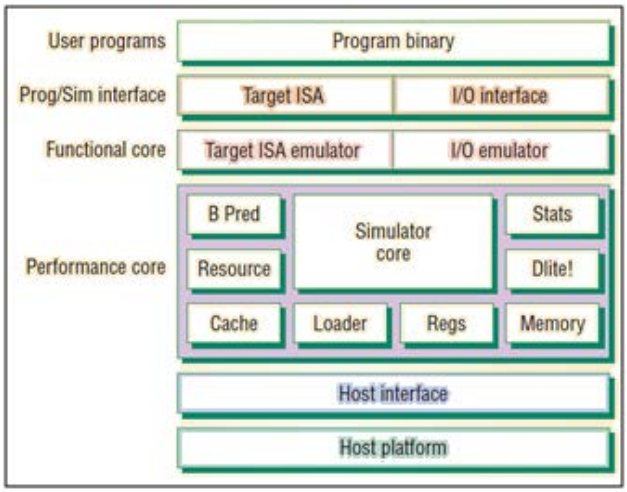

Figure 7: Simple-Scalar simulator software architecture [20].

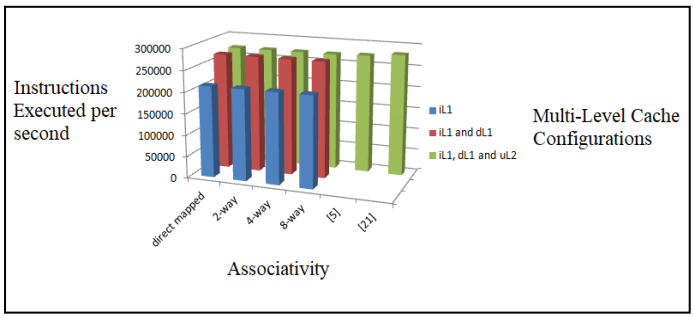

Figure 8: Instructions executed (per second) per multi-level cache configuration

Figure 8 compares the number of instructions executed by each multi-level cache configuration. It can easily be analyzed that the change in the number of instructions executed does not vary at all with the change in the associativity of cache, however, it increases when we introduce a L1 data cache into the configuration. Moreover, there is a significantly low change in the number when we introduce a L2 unified cache. This is due to the fact that a certain amount of parallelization is introduced when multiple levels of caches are introduced. When there is a miss in L1 cache then the search goes onto L2 cache and a new instruction can be executed in L1 cache. Regarding the number of instructions executed, our results is similar with the results of [19] and [20].

7. Concluding Remarks

This paper presented the result of an empirical study and performance analysis of a multilevel cache with customized configurations. The research questions focused on how multi-level caches, with different factors such as associativity, could impact the performance of a CPU and thus the system. We surveyed and analyzed several cache simulation studies to configure our needs. Futher, we examined the impact of key factors, such as access time, cycle time, area on chip, the total number of instructions executed, total number of hits and miss-rates. The selected tools helped us to simulate cache and in depth understanding the design factors. We compared the obtained results with those reported in the literature. In most cases, the results were comparable, and in some cases slight improved were achieved.

Bibliography

1. Hill M.D, and Smith A.J. Evaluating Associativity in CPU Caches. In: IEEE Transactions on Computer, 1989.
2. Arjun Malik A., Bhatia M.S, Wu P., Zhe Qi, Cache Coherency Case Study: Cache Pipeline, Multilevel, Hierarchical, Semester Project, Dept. Computer Science, BGHI, Ohio, 2017.
3. Duska, B. M., Marwood D, and Feeley M. J. The Measured Access Characteristics of World-Wide-Web Client Proxy Caches. USENIX Symposium on IT and Systems. Vol. 997. 1997.
4. Murlimanohar N, Balasubramonium R, Jouppi N.P. CACTI 6.0: A Tool to Model Large Caches. HP Laboratories, 2009
5. Todd Austin, SimpleScalar LLC, www.simplescalar.com

6. S. Przybylski, M. Horowitz, J. Hennessey. Characteristics of performance-optimal multi-level cache hierarchies. ACM SIGARCH Computer Architecture news, June, 1989
7. Conte T.M., Hirsch M.A., Hwu W. Combining Trace Sampling with Single Pass Methods for Efficient Cache Simulation. In: IEEE Transactions on Computers, 1998
8. Sugumar R, Abraham S. Set Associative Cache Simulation Using Generalized Binomial Trees. In: ACM Transaction on Computer Systems, 2005.
9. Hardy D, Puaut I. WCET Analysis of Multi-level Non-inclusive Set-Associative Instruction Caches. IEEE, December, 2008
10. Srikantaiah S, Kultursay E, Zhang T, Kandemir M, Irwin M.J, Xie Y. MorphCache: A Reconfigurable Adaptive Multi-level Cache hierarchy. IEEE, February, 2011
11. Kecheng Ji, Ling M, Wang Q, Shi L., Pan J., AFEC An Analytical Framework for Evaluating Cache performance in out of order processors. In 2017 Design, Automation & Test in Europe Conf. & Expo.
12. Yavits L., Morad A., Ginosar, Cache Hierarchy Optimization. In IEEE COMPUTER ARCHITECTURE LETTERS, VOL.13, NO. 2, 2014.
13. Waldspurger C, Saemundson T., Ahmad I. and Park N., Cache Modeling and Optimization using Miniature Simulations. In 2017 USENIX Annual Technical Conference (USENIX ATC '17).
14. Hachem J, Karamchandani N, Diggavi S.N. Coded Caching for Multi-level Popularity and Access. In: IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. 63, NO. 5, 2017.
15. Maeda R, Cai Q, Xu1 J, Wang Z, and Tian Z., Fast and Accurate Exploration of Multi-Level Caches Using Hierarchical Reuse Distance. In 2017 IEEE conf on High Performance Computer Architecture.
16. Zhang Z, Guo Z and Koutsoukos X. HandlingWrite Backs in Multi-Level Cache Analysis for WCET Estimation. In 25th Conf. on Real-Time Networks and Systems, 2017, Grenoble, France.
17. Shatnawi A and Alsaedeen M. Reducing the second-level cache conflict misses using a set folding technique. In: The Journal of Supercomputing 2018, volume 74, pp 970–993.
18. Gunadi E, Mikko H. Lipasti, Cache Pipelining with Partial Operand Knowledge. 2004.
19. Ozdemir S., Chun Ku J., Mallik A., Memik G, Ismail Y., Variable Latency Caches for Nanoscale Processors. Technical Report, NWU-EECS-06-16, October 23, 2006.
20. Schaum's Outline of Theory and Problems of Computer Architecture. The McGraw-Hill Companies Inc. Indian Special Edition 2009
21. Alan Jay Smith. Cache Memories. Computing Surveys, ACM, September 1982