International Journal of Humanities and Social Science Invention (IJHSSI)
ISSN (Online): 2319 – 7722, ISSN (Print): 2319 – 7714
www.ijhssi.org ||Volume 8 Issue 05 Ser. II || May 2019 || PP 39-46

A Analysis Of Low Power Design Techniques For Last Level Caches Using Way Filter Technique

Dr. Nagesh K N[1], ShashikiranR[2] , Manjunatha M[3]
1. HOD and Professor Department of ECE, Nagarjuna College of Engineering and Technology, Bangalore
2. Assistant Professor Department of ECE,Nagarjuna College of Engineering and Technology, Bangalore.
3. IV Sem, M.Tech, Department of ECE, Nagarjuna College of Engineering and Technology, Bangalore.
Corresponding Author: Manjunatha M

ABSTRACT: *Last-level caches (LLCs) help improve performance but suffer from energy overhead because of their large sizes. An effective solution to this problem is to selectively power down several cache ways, which, however, reduces cache associativity and performance and thus limits its effectiveness in reducing energy consumption. To overcome this limitation, we propose a new cache architecture that can logically increase cache associativity of way-powered-down LLCs. Our proposed scheme is designed to be dynamic in activating an appropriate number of cache ways in order to eliminate the need for static profiling to determine an energy-optimized cache configuration. The experimental results show that our proposed dynamic scheme reduces the energy consumption of LLCs by 60% on single- and dual-core systems, respectively, compared with the best performing conventional static cache configuration. The overall system energy consumption including CPU, L3 cache, and DRAM is reduced by 3.64% on Pentium Quad core systems. The project consists of three different tasks: 1) Design - Designing a low-power cache memory (instruction or data) at the abstract level after literature research; 2) Code - Writing a simulation program on top of a simulator (e.g., Simple scalar); and 3) Test - Running a test program to evaluate the low-power cache memory by using performance metrics.*
Index Terms— *Cache, energy, multicore system.*

Date of Submission: 06-05-2019 Date of acceptance:21-05-2019

I. INTRODUCTION

Last-level caches (LLCs) play important roles of improving performance and reducing energy consumption by filtering out costly accesses to the memory system. The size of LLC is getting larger to support multicore systems better, which run many programs simultaneously. Thus, the LLC consumes more energy when many cores are involved. A large LLC that exceeds program demand capacity generally increases energy consumption without improving performance. Because programs demand different LLC capacities [1] and SRAM LLCs consume large leakage energy, several techniques have been proposed to reduce leakage energy consumption of LLCs. An effective and natural solution is to selectively power down some LLC ways when LLC capacity exceeds a needed capacity by programs, which eliminates leakage energy consumption in these powered down LLC ways [2]. However, the reduced associativity of way-powered-down LLC decreases performance due to increased LLC conflict misses, which may result in actual increase in energy consumption.way-filtering (WF)-based logical–associative LLC architecture to reduce the energy consumption of LLCs. This architecture logically increases the associativity of LLCs when one to three cache ways are activated, and thus improves performance and reduces energy consumption. To further decrease tag way energy consumption, we utilize a partial tag-based WF scheme. In addition, a sequential logical way accessing and indexing scheme is proposed to support multiple LLC logical way accesses when multiple logical way hits occur in one physical way using the partial tag-based way filter as show in fig below.

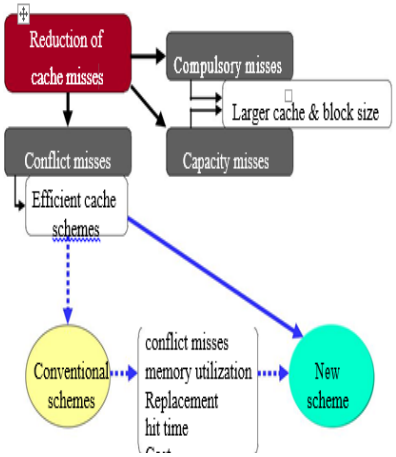

Figure 1. Three types of cache misses and a low-power cache scheme

To make our proposed WF-based logical–associative LLC architecture to be practical, we propose a dynamic resizing algorithm to eliminate the need for static cache profiling to determine an energy-optimized LLC configuration. "Energy-optimized configuration" means the configuration that consumes the least energy. Starting from one LLC data way, our dynamic resizing algorithm activates more or fewer LLC data ways using the approximate standard deviation of cache misses of LLC logical sets as a metric for measuring cache way demand. A logical set is a set of cache lines that use the same index in logical ways. A logical way is an internal cache way that is divided from a cache way.

II. OBJECTIVE

In general, conflict misses are very critical for a small cache size, especially level-one on-chip cache memory, and directly affect the power consumption, system performance, and costs. Therefore, the goal of the project is mainly on designing an efficient cache memory to reduce conflict misses and power consumption.

III. METHODOLOGY

There are five main factors to design a cache memory [3]. Those are the cache size, block size, mapping function, replacement policy, and writing policy. In this paper, we focus on small on-chip cache sizes (e.g., 32KB or 64 KB) with the popular block sizes (e.g., 32bytes or 64 bytes) and include one more factor, power consumption, to design a low-power cache memory for embedded systems. Therefore, there are four factors to design a cache memory, except cache and block sizes: 1) Design a mapping function; 2) Design a replacement policy; 3) Design a writing policy; and 4) Design a low-power cache memory. Since there are many cache specifications to determine, we recommend students to work as a team, 2 to 3 students per team, to discuss many possible topics.

Design a mapping function

For the first step, each team is required to design a mapping function to access the cache memory in an efficient way.For example, Figure 2 shows two different mapping functions: 2-way set-associative (conventional) and 2-way skewed-associative (more efficient one). Each scheme has two blocks and each block has its own mapping function. For Figure 2-a, the mapping functions F_0 and F_1 are the same. Therefore, if three instructions (A0, A1, and A2) access to the same location in the block 0 and block 1, there should be a conflict since they are located in only two blocks. Meanwhile, if the mapping functions F0 and F1 are different like Figure 2-b, the conflict can be resolved since three instructions can be placed into three different locations in the Block 1. Therefore, the mapping function is an important factor to reduce cache misses. Each team can design any kinds of mapping function to reduce conflict misses by dispersing instructions in a block.

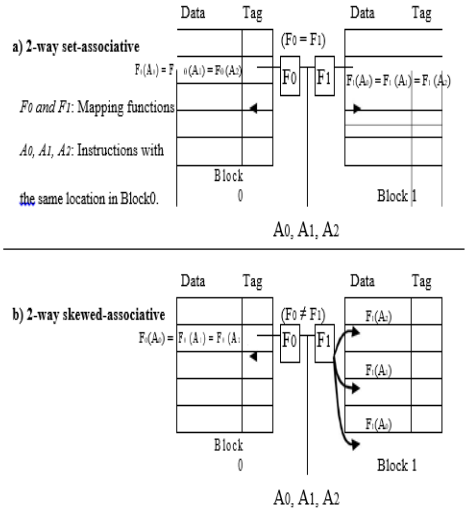

Figure 2. Mapping functions for 2-way set-associative and 2-way skewed-associative.

IV. RELATED WORK AND BACKGROUND

Many circuit-level techniques for reducing the leakage energy of the cache memory have been proposed. The gated-Vdd proposed by Powell et al. [4] is used in many cache designs [18], but it incurs data loss. The drowsy cache scheme [16] is proposed to reduce leakage energy in unused individual cache lines without data loss. The DRG-cache scheme [17] is proposed to reduce leakage energy of a cache memory without data loss. Its hardware complexity is lower than that of the drowsy cache, but its energy reduction is smaller. However, it is difficult for these two techniques to be applied to real circuits due to process variations, which can make low-voltage SRAM cells faulty [30]. Many architectural studies have been conducted to reduce the leakage energy of the cache memory. The most well-known technique is the selective cache ways [2], which was developed to decrease dynamic energy but is also effective in reducing leakage energy. This technique selectively disables a subset of cache ways to reduce cache energy, as shown in Fig. 1(a). This approach is quite effective because many programs do not require an entire cache capacity [1]. This fact has been exploited by many studies to reduce energy consumption or to increase performance. Many researchers tried to reduce the number of ways with little performance degradation [8], [9]. Determining how many cache ways are required is very important. Hwang and Li [8] distinguished repeated and fresh misses to determine the appropriate associativity in a cache set. A dynamic cache resizing technique for multicore systems has recently been proposed [9]. Qureshi and Patt [10] used utility monitors to compute an appropriate cache size when cache resizing is performed. We implement the proposed cache size estimation scheme to compare it with our proposed cache architecture.

V. PROPOSED WAY-FILTERING-BASED LOGICAL–ASSOCIATIVE CACHE ARCHITECTURE

To reduce the energy consumption in the tag ways of LLCs, we apply a partial tag matching scheme [11], [12]. It extracts a few bits from the tag bits to early identify a cache miss. Fig 3. shows our proposed partial tag-based way filter. A partial tag consists of a few least significant bits from the original tag bits and a few most significant bits from the index bits. Because the cache lines in a cache way are logically divided, the most significant bits (3 bits for eight logical ways) from the index bits are used as part of a partial tag. Its because the number of logical cache sets becomes smaller than that of cache sets when logical cache ways is applied. A partial tag-based way filter is allocated to each cache way, and it is powered down when its corresponding cache way is turned OFF. Experimentally, we find that a 4-bit partial tag and eight logical ways show optimized results. The detailed experimental results are presented.

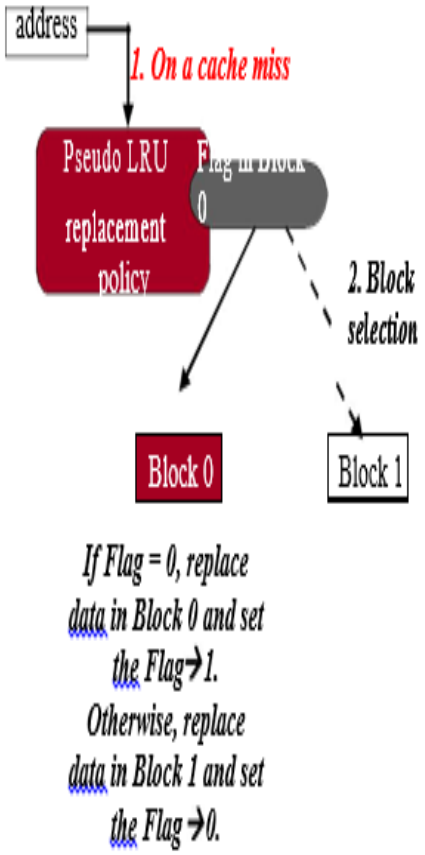

Figure 3. Replacement policy for 2-way skewed-associative.

There are two types of the writing policy: write-hit policy and write-miss policy. For the write-hit policy, the write-back policy is popular since it updates data only in the cache memory until it is replaced with the data from the lower-level memory (memory). If it is replaced because of a cache miss, the data should be updated to the memory before it is replaced in the cache memory. It works well to reduce memory access time, which is much slower than the cache memory, compared to the write-through policy. The write-through policy updates the cache memory and memory at the same time for any cache write-hits. It would take more time compared to the write-back policy since it should access the memory for every write-hit case.For the write-miss policy, there are also two general policies, such as the write-allocate (to update the cache memory first and memory later) and the write-no-allocate (to update memory only) for a write-miss. Each team can design write policies based on the conventional write policies to accommodate future references effectively.Design a low-power cache memoryFigure 4 shows hardware-complexity comparison between two cache memories since cost is also an important factor for cache memory design. In Figure 4, the cost of 2-way skewed-associative would be more expensive than 2-way set-associative because 2-way skewed-associative uses xor mapping functions, which is more complex than 2-way set-associative. However, since the xor mapping functions can reduce cache misses effectively, there can be a tradeoff between two cache memories regarding the performance and cost.To reduce power consumption for a cache memory, it has been necessary to reduce the frequency of memory accesses by developing techniques such as line buffering. Some research has indicated that a small buffer line between CPU and an L1 (Level-one) cache memory helpsin reducing the accesses to the L1 cache and thus reduces the energy dissipation in the L1 cache.

Contents		2-way set-associative	2-way skewed-associative
Logic and Indexing	Replacement	LRU, etc.	PLRU, etc.
	Indexing	Lower part of address	XOR mapping
Hardware	Blocks	2	2
	Flag	Y (Block 0)	Y (Block 0)
	Block design	Classical design	Classical design + XOR gates (mapping)
	Counter	N	N
Access time		Same	Same (slightly increased by XOR gates)
Hardware complexity		Same	Almost Same (Only several XOR gates are added to the 2-way set-associative)
Cache miss ratio		High	Medium

Figure 4. Comparison hardware Complexity for 2-way cache memories.

For example, the CoC (Caching on Cache) is a small-sized sub-cache of the L1 cache [10]. The CoC is accessed first for every memory reference: If the reference is a hit, the lines in the L1 cache are disabled (no access to L1 cache). Otherwise, the L1 cache is accessed normally. The access operation involves maintaining a tag and data of the cache lines and also adds latency to the normal cache access. For another example, the Filter cache is also a small cache between the CPU and the conventional L1 cache (the conventional L1 cache used to be treated as a L2 cache)The Filter cache contains data with high hit probability based on locality. Therefore, a miss on the filter cache directs the references to the L2 cache, which used to be a conventional L1 cache. The power savings are earned by maintaining a small-size L1 cache (Filter cache) instead of a regular L1 cache. These techniques require additional data and tag arrays that consume power with every reference. Each team can design a low-power cache memory like the above examples.

Code Procedure

Since the advanced computer architecture class is a graduate-level course, programming languages like C/C++ or VHDL/Verilog would be prerequisites for the class in general [4].In the previous section, each team could design the mapping function, replacement policy, writing policy, and low-power cache memory for their own purpose. The next step is to write a code for their cache memory. In other words, after students write a code according to the procedures, they should port the code into a simulator like SimpleScalar. Figure 5 shows five SimpleScalar standard simulator models as an example.

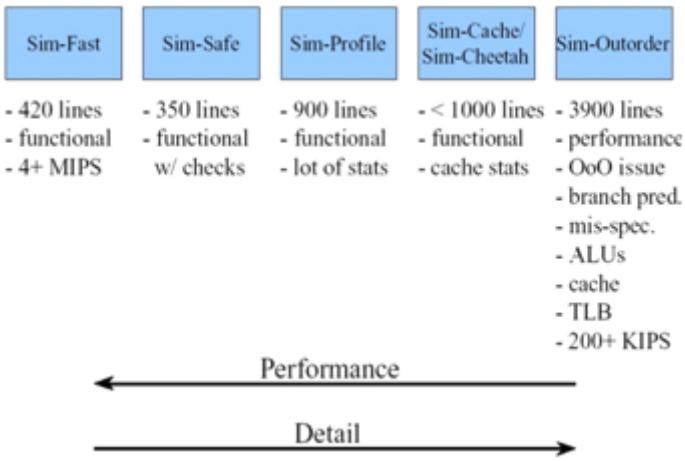

Figure 5. Simple Scalar standard models.

Test Procedure

After students complete the design and code procedures, they need to test their architecture with the benchmark program Figure 6 shows the procedure how to test the cache memory with benchmark programs. The benchmark programs should be compiled and linked to produce the binary, which is the real input for the simulator. Since the simulator is a virtual architecture, it has its own instruction sets; and it is necessary to compile the benchmark programs with the compiler provided by the SimpleScalar to run Benchmark executables (e.g., compiled binaries) [9].

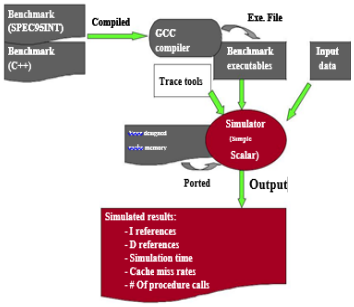

Figure 6. Test procedure with benchmark programs.

After you create the input binaries, you can implement the simulator with the input data by using the following command line in Figure 6 shows the command line with many options, such as a level-one data cache (dl1), 32 lines in a cache (32), associativity (1 as a direct-mapped cache), replacement policy (1), the binary input program (/bin/test.ss), etc. Each team can add more options after modifying some procedures in the SimpleScalarmodels.After you complete the simulation with benchmark programs, like the Figure 7, you can get the simulation results, such as I references (# of instructions), cache miss rates, etc. With the results, you can evaluate your cache memory by using some metrics, such as cache miss rate, execution time, IPC (Instructions per cycle), power, etc. Those metrics are [1]:

- CPU Execution time = IC × Effective CPI × t – (1)
- Effective CPI = Ideal CPI + Average memory stalls per instruction – (2)
- AMAT = Hit time + Miss rate × Miss Penalty – (3)

** IC (Instruction Count), CPI (Cycle Per Instruction), AMAT (Average Memory Access Time).

VI. EXPERIMENTAL RESULTS

Our CPU baseline model is the ARM Cortex-A8 [22], which is widely used in high-performance embedded systems. The L2 cache in our experiments is the LLC. In L2 caches, tag ways are accessed first and data ways are accessed next [22]. When a partial multihit occurs, only the tag access delay is added for sequential tag probes. All caches use a write–back policy, and their replacement policies are LRU. The technology node is 32 nm. Fig 7 lists the configuration of our baseline processor.

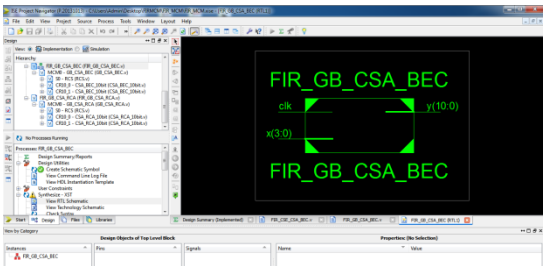

Fig 7: Results of baseline processor.

WF-based dynamic logical–associative cache is a standard cache with some additional components so that we use CACTI to model our proposed cache. The core energy is calculated using McPAT [23]. We modified the McPAT input file generation script in Sniper to model the ARM Cortex-A8 architecture. The DRAM energy is modeled by DRAMPower [28]. DRAMPower requires memory transactions to model DRAM energy consumption. We modified Sniper to output a DRAM memory-transaction file and used it as the input for DRAMPower. "Power down" mode is used to reduce idle power consumption, which makes DRAM ranks nap when there is no outstanding memory requests in the memory controller. This technique is used in many real systems [33]. We consider the energy consumption of both processor and DRAM to demonstrate the real benefit of our proposed resizable WF-based dynamic logical–associative cache scheme.

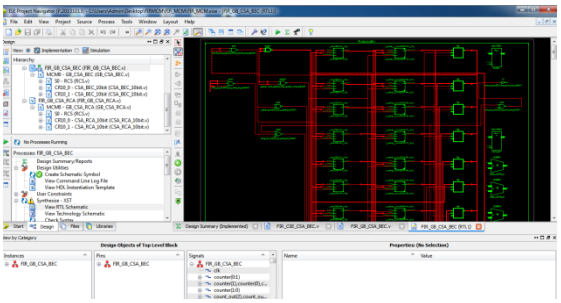

Fig 8: Energy consumption with DRAM

We measure the accuracy of the partial tag-based way filters with 3-, 4-, and 5-bit tag sizes for eight and four logical ways of WF-based logical–associative caches with a 64-kbyte capacity, which is the size of one cache way in our baseline LLC. Table IV lists the results of the partial tag hits. The oneway hits increase with lower associativity and larger partial tag size. Multiway hits can occur, but most of the hits are two-way hits. Moreover, accessing the tag ways only once is possible if the first cache probe hits. Thus, a multiway hit penalty is not critical. The 3-bit partial tag shows a relatively low partial one-way hit ratio. The 5-bit partial tag shows a higher partial one-way hit ratio than the 4-bit partial tag, but their actual execution times are almost the same. The global LLC access ratio is low, and most of them are partial one-way hits. Therefore, we choose 4 bits as the partial tag size in our experiments. We select eight as default logical associativity because some benchmarks show notable performance increase over four (crafty, eon, and mesa show 20.3%, 44.2%, and 15.8% performance increases, respectively).

VII. CONCLUSION

There have been so many software tools developed to teach computer architecture classes. Traditionally, those tools have many options to select for any proper operations or consist of lengthy lines of code to figure out. Therefore, students are required to figure out the options first and then learn the proper operations. In addition, since the tools used to have limited functions to operate, it is difficult to design a new function logic with the tools. Therefore, those tools let students understand only the limited operations instead of creative design since they lack experience of the designing process.

FUTURE SCOPE

we believe that students could learn fundamental concepts and the design process clearly for the advanced computer architecture class and gain confidence in the area of low-power cache memory design.

REFERENCES

[1]. A. Jaleel. Memory Characterization of Workloads Using Instrumentation-Driven Simulation, Sep. 2, 2016. [Online]. Available: http://www.jaleels.org/ajaleel/workload/SPECanalysis.pdf

[2]. D. H. Albonesi, "Selective cache ways: On-demand cache resource allocation," in Proc. MICRO, Nov. 1999, pp. 248–259.

[3]. S.-H. Yang, M. D. Powell, B. Falsafi, and T. N. Vijaykumar, "Exploiting choice in resizable cache design to optimize deep-submicron processor energy-delay," in Proc. High Perform. Comput. Archit., Feb. 2002, pp. 151–161.

[4]. M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar, "Gated-Vdd : A circuit technique to reduce leakage in deep-submicron cache memories," in Proc. Int. Symp. Low Power Electron. Design, Jul. 2000, pp. 90–95.

[5]. J. Lee and S. Kim, "Filter data cache: An energy-efficient small L0 data cache architecture driven by miss cost reduction," IEEE Trans. Comput., vol. 64, no. 7, pp. 1927–1939, Jul. 2015.

[6]. B. Calder, D. Grunwald, and J. Emer, "Predictive sequential associative cache," in Proc. High Perform. Comput. Archit., Feb. 1996, pp. 244–253. [7] M. Ghosh, E. Ozer, S. Ford, S. Biles, and H.-H. S. Lee, "Way guard: A segmented counting bloom filter approach to reducing energy for setassociative caches," in Proc. Int. Symp. Low Power Electron. Design, Aug. 2009, pp. 165–170.

[7]. Y.-S. Hwang and J.-J. Li, "Snug set-associative caches: Reducing leakage power of instruction and data caches with no performance penalties," ACM Trans. Archit. Code Optim., vol. 4, no. 1, Mar. 2007, Art. no. 6.

[8]. D. Kadjo, H. Kim, P. Gratz, J. Hu, and R. Ayoub, "Power gating with block migration in chip-multiprocessor last-level caches," in Proc. Int. Conf. Comput. Design, Oct. 2013, pp. 93–99.

[9]. M. K. Qureshi and Y. N. Patt, "Utility-based cache partitioning: A lowoverhead, high-performance, runtime mechanism to partition shared caches," in Proc. MICRO, 2006, pp. 423–432.

[10]. R. Min, Z. Xu, Y. Hu, and W. B. Jone, "Partial tag comparison: A new technology for power-efficient set-associative cache designs," in Proc. Int. Conf. VLSI Design, Jan. 2004, pp. 183–188.

[11]. L. Liu, "Cache designs with partial address matching," in Proc. MICRO, Nov. 1994, pp. 128–136.

[12]. F. M. Sleiman, R. G. Dreslinski, and T. F. Wenisch, "Embedded way prediction for last-level caches," in Proc. Int. Conf. Comput. Design, Sep./Oct. 2012, pp. 167–174.

[13]. A. Agarwal, J. Hennessy, and M. Horowitz, "Cache performance of operating system and multiprogramming workloads," ACM Trans. Comput. Syst., vol. 6, no. 4, pp. 393–431, Nov. 1988.

[14]. A. Agarwal and S. D. Pudar, "Column-associative caches: A technique for reducing the miss rate of direct-mapped caches," in Proc. Int. Symp. Comput. Archit., May 1993, pp. 179–190. [16] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge, "Drowsy caches: Simple techniques for reducing leakage power," in Proc. Int. Symp. Comput. Archit., May 2002, pp. 148–157.

[15]. A. Agarwal, H. Li, and K. Roy, "DRG-cache: A data retention gatedground cache for low power," in Proc. Design Autom. Conf., Jun. 2002, pp. 473–478.

[16]. M. Sato, R. Egawa, H. Takizawa, and H. Kobayashi, "A voting-based working set assessment scheme for dynamic cache resizing mechanisms," in Proc. Int. Conf. Comput. Design, Oct. 2010, pp. 98–105.

[17]. SPEC CPU2000, Sep. 2, 2016. [Online]. Available: http://www.spec.org/cpu2000/

[18]. T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in Proc. Int. Conf. High Perform. Comput.,Netw., Storage Anal., Nov. 2011, pp. 1–12.

[19]. N. Muralimanohar et al., "Cacti 6.0: A tool to understand large caches," Hewlett Packard Lab., Univ. Utah, Salt Lake City, UT, USA, Tech. Rep., 2007. [22] ARM Cortex-A8, accessed on Feb. 3, 2015. [Online]. Available: http://www.7-cpu.com/cpu/Cortex-A8.html

[20]. S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in Proc. MICRO, Dec. 2009, pp. 469–480.

[21]. M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in Proc. IEEE Annu. Workshop Workload Characterization, Dec. 2001, pp. 3–14. [25] H. Patil, R. Cohn, M. Charney, R. Kapoor, A. Sun, and A. Karunanidhi, "PinPoints: Pinpointing representative portions of large Intel Itanium programs with dynamic instrumentation," in Proc. 37th Annu. IEEE/ACM Int. Symp. Microarchitecture, Portland, OR, USA, Dec. 2004. [26] B. Batson and T. N. Vijaykumar, "Reactive-associative caches," in Proc. Int. Conf. Parallel Archit. Compilation Techn., Sep. 2001, pp. 49–60. [27] (2008). Mobile Intel Atom Processor N270 Single Core, accessed on Feb. 3, 2015. [Online]. Available: http://www.intel.com/ content/dam/doc/datasheet/mobile-atom-n270-single-core-datasheet-.pdf

[22]. Drampower: Open-Source Dram Power & Energy Estimation Tool, accessed on Feb. 3, 2015. [Online]. Available: http://www. es.ele.tue.nl/drampower/

[23]. W. Wang and T. Dey. (2011). A Survey on ARM Cortex A Processors, accessed on Feb. 3, 2015. [Online]. Available: http://www.cs. virginia.edu/~skadron/cs8535_s11/ARM_Cortex.pdf

[24]. A. Sasan, K. Amiri, H. Homayoun, A. M. Eltawil, and F. J. Kurdahi, "Variation trained drowsy cache (VTD-cache): A history trained variation aware drowsy cache for fine grain voltage scaling," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 20, no. 4, pp. 630–642, Apr. 2012. [31] S. Kim, "Reducing area overhead for error-protecting large L2/L3 caches," IEEE Trans. Comput., vol. 58, no. 3, pp. 300–310, Mar. 2009. [32] S. Tomar, S. Kim, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "Use of local memory for efficient java execution," in Proc. Int. Conf. Comput. Design, Sep. 2001, pp. 468–473.

[25]. SK Hynix. DDR3 Device Operation, accessed on Sep. 3, 2015. [Online]. Available: https://www.skhynix.com/product/filedata/fileDownload.do? seq=2354

[26]. D. A. Patterson and J. L. Hennessy, Computer Organization and Design, 4th ed. San Mateo, CA, USA: Morgan Kaufmann, Nov. 2008.

Manjunatha M" A Analysis Of Low Power Design Techniques For Last Level Caches Using Way Filter Technique" International Journal of Humanities and Social Science Invention (IJHSSI), vol. 08, no. 5, 2019, pp.39-46