Bulletin of Electrical Engineering and Informatics
Vol. 10, No. 5, October 2021, pp. 2910~2920
ISSN: 2302-9285, DOI: 10.11591/eei.v10i5.2459

Modeling cache performance for embedded systems

V. C. Chijindu, O. K. Ugwueze, C. C. Udeze, M. A. Ahaneku, J. N. Eneh, O. M. Ezeja, E. C. Anoliefo
Department of Electronic Engineering, Faculty of Engineering, University of Nigeria, Nsukka, Nigeria

Article Info

Article history:

Received Apr 20, 2020
Revised Apr 29, 2021
Accepted Jul 23, 2021

Keywords:

Cache designs
Cache memory
Cache model
Cache performance
Embedded systems
Reuse distance

ABSTRACT

This paper presents a cache performance model for embedded systems. The need for efficient cache design in embedded systems has led to the exploration of various methods of design for optimal cache configurations for embedded processor. Better users' experiences are realized by improving performance parameters of embedded systems. This work presents a cache hit rate estimation model for embedded systems that can be used to explore optimal cache configurations using Bourneli's binomial cumulative probability based on application of reuse distance profiles. The model presented was evaluated using three mibench benchmarks which are bitcount, basicmath and FFT for 4kb, 8kb, 16kb, 32kb and 64kb sizes of cache under 2-way, 4-ways, 8-ways and 16-ways set associative configurations, all using least recently-used (LRU) replacement policy. The results were compared with the results obtained using sim-cheetah from simplescalar simulators suite. The mean errors for bitcount, basicmath, and FFT benchmarks are 0.0263%, 2.4476%, and 1.9000% respectively. Therefore, the mean error for the three benchmarks is equal to 1.4579%. The margin of errors in the results was below 5% and within the acceptable limits showing that the model can be used to estimate hit rates of cache and to explore cache design options.

Corresponding Author:

O. K. Ugwueze
Department of Electronic Engineering
University of Nigeria, Nsukka
Enugu State, Nigeria
Email: kingsley.ugwueze@unn.edu.ng

1. INTRODUCTION

The need for cache memory arises as a result of the speed of processor being higher than that of the main memory. It is worthy to note that the memory wall problem [1], [2] in general purpose computer also affects embedded systems. The problem is that at a point an increase in processor speed does not contribute much to performance of computer systems [3]. This is because the net increase in performance of a computer system do not depend on processor alone but also dependent on other factors such as memory speed, input/ output device and the bus configuration. Moreover, increasing processor speed means increasing the clock rate which has an adverse effect in term of power dissipation [4]. The major setback in computing system performance is the improvement in the speed of memory devices such as DRAM [5] which is very slow compared to processor. According to Wilkes [3], for two decades now, there have not being any major improvement recorded in the area of memory speed. It means that processor has to waste most of it clock cycle waiting to be serviced by the main memory. In effect, this memory speed limits the performance of computer systems drastically. This has led to researches on how to improve the memory speed in order increase the speed of computer systems. The formal way of closing this speed gap is by using cache memory between the main memory and processor. In this quest, designers started using cache memory which is

smaller by faster memory device made from static random access memory (SRAM) technology to copies locations of memory in other to service the processor faster on request [6]. A typical embedded systems level one L_1, data cache memory is found in Arm Cortex A7 [7]. It has only two levels of cache memory but with split data and instruction cache. The level one L_1, data cache organization is set associative with least recently used replacement policy. When a processor requests a memory location, that location is first search for in the cache and if it is not in cache, the location will be fetched from main memory. This main memory location is first copied into the cache before loading to the processor. Next time when this same memory location is requested, it will be serviced from the cache thereby hiding the real speed or reducing the latency of the main memory. This process helps to improve the speed of memory systems.

The idea behind the success of cache memory is locality of reference in a program [8]-[11]. Program execution usually favours portion of the main memory within a short duration of time. According to Eklov *et al.* [12], the factors that contribute to this locality patterns are sequential execution of instructions, loops in a program, and nature of data items stored. First, principle of locality states that program tends to reuses memory location in which it had accessed and that is called temporary locality [13], [14]. Secondly, it states that it is also likely that location of the memory close to the recently referenced memory location may be referenced in near future and it is called spatial locality [15]. These two properties of locality are what drive the effectiveness of cache memory. The block of main memory which is referenced is copied into the cache line following the mapping procedures. At the time that the cache is full any memory location requested by the processor which is not in cache means that an already existing memory block in the cache have to be evicted so that the recently referenced block will be copied into the cache. The procedure in which a block of main memory already in the cache is selected for eviction from the cache and to be replaced by the most recent request is called cache replacement policy [16]. The major replacement policy that are mostly used are least recently used (LRU), random and first-in first-out (FIFO) replacement policies. The choice of replacement policies used by computer architecture is dependent on the performance optimization requirement and cost. Another important aspect of cache memory design is its degree of associativity with the main memory. The procedure in which the block of main memory is mapped into the line of the cache is called cache associative [17]. Some of the well-known standard caches mapping techniques are set associative, direct mapping and fully associative mapping [18]. Cache associative is an important aspect of cache performance criteria. So, it is worthy to note that cache hit rate and miss rate of cache is dependent on the type of associative of the cache.

In order to calculate or predict the locality and performance of cache memory analytically through any performance parameter like cache hit rate, latency and effectiveness, a metric that is microarchitectural independent and also representative of the workload is required. Reuse distance and reuse time [19], [20] are two most popular metrics that is used in prediction of cache performance. Reuse distance is the number of unique intervening memory accesses between the use and the reuse of a particular memory location while reuse time is the number of total or absolute memory accesses between the use and reuse of such memory location. During program execution, processor makes series of reference to various memory locations that is required for successful execution of the program. The flow of this memory location reference is called memory reference stream. As the processor makes this reference, a collection of the memory location referenced is called memory profile [6] which is very important input parameter in cache analysis. Memory profile is useful as stack distance or reuse distance of a memory stream is obtained from it. The locality of cache is determined through the analysis of program memory reference stream. If a memory reference stream exhibits a significant temporary locality (e.g., once accessed, references to the same address location is likely in near future) or spatial locality (e.g., once accessed, references to the neighbouring address locations is likely in near future), the cache hit rate hence overall performance will be high [6]. According to Zhong [21], the smaller the reuse distance of memory location of an application, the more the application obeys the principle of locality. This give rise to overall increases in cache performance but if the reuse distances are large there is high probability that the application will yield low hit rate in cache thereby reducing the cache performance. In this study, the metric used was reuse distance because it describes well the locality of memory accesses and is closely related to the behavior of LRU policy which allowed us to assess how well the cache is utilized. Furthermore, reuse distance is machine-independent and deterministic, making it an ideal metric for used in performance modeling.

The earlier background for analytical cache model is from the work of Mattson *et al.* [8], when they developed a stack counting algorithms that processed accesses of an applications to memory locations and their reuse as distance of the first access of that memory location in the stack to the its present access called stack distance or reuse distance. Their interest was not to calculate or model cache behavior but to analyze the locality of an application, as a result they did not come up with any cache model but their result is background to this present study. According to Enbody and Brehob [22], they proposed an analytical cache framework based on stack distance distribution to describe the behavior of cache. They developed model to describe the locality of a reference stream and another one used to quantify cache locality and behaviour. It is

important to the present study because the cache model proposed in this study is an extension of their second model. In the work of Pan and Jonsson [20], they proposed an analytical cache framework based on absolute reuse time distribution to describe the behavior of cache.

The study used Pin [23] to trace the workload which are SPEC CPU2006 [24] benchmarks to get the reuse time which is feed into their model for evaluation. Their study pointed out that profiling stack distance of an application incurs much overhead so they resorted to low overhead metrics of absolute reuse distance or reuse time. Beckman and Sanchez [25] also proposed a new probabilistic cache model based on absolute reuse distance for high performance replacement policies. The model used reuse times which were collected using hardware monitor and models replacement policies as abstract ranking functions. Their model is an age-based model which comprises of age, eviction and hit distribution models. Another study by Chen *et al.* [10] which focused on the optimal multi-level cache design used reuse distance metric to model cache performance. In their work, they predicted the miss rate of multi-level fully associative cache with LRU replacement policy using reuse distance metric. They assumed that fully associative LRU cache can be used as representative of set associative cache as there is no significant difference in their performance giving the same cache size. Just like the present study, they used Pintool to collect metrics and also compared their results with results from Simplescalar simulation; their average error is 0.71% (L2) and 1.1% (L3). Gysi *et al.* [26] proposed an analytical cache model algorithm that predict the miss and hit rate of fully associative with LRU cache. They called their algorithm Haystack, and it make used of calculated program reuse distance to predict performance just like the present study. The study used Polybench in evaluation of their model and compared results with results from Uniprocessor simulator. They also validated their results using measures results from PAPI [27]. The evaluation errors in their model were within 0.6% to the measured results.

This study will serve as useful tool for cache performance analysis as it will help computer architects, researchers and students to have insight into cache behavior under different configurations. It will also help them in design decision for system optimization and management. Equipped with proper understanding of program locality and how to extract reuse distance from memory profile of a program, prediction of cache performance will now become easier. The aim of this present study is to presents a model of cache performance for embedded systems and the specifics objectives include:To develop a mathematical cache hit rate estimation model; To characterize the locality of embedded system workloads; To evaluate the model using the metrics from characterized embedded system workloads; and To compare the behaviour of the model with that of standard cache simulator by applying the same characterized embedded workloads and cache configurations parameters.

2. PROPOSED CACHE HIT RATE ESTIMATION MODEL

To model cache behaviour, the aim is to use reuse distance obtained using MICA pintool to generate our model. This means that given a memory trace, ***T***, for every reuse distance, 'd' one would want to know the probability of cache hit. It is obvious that in set associative cache with LRU replacement policy that the reuses of memory block for a distance that is less than the associativity of the cache is a hit. This is because for any memory block to be replaced in cache there must be up to '***A***' number of distinct access to that set that contain memory block where '***A***' is the degree of associative of the cache. Therefore, the hit ratio of all the accesses with reuse distance ***d*** which is less than or equal to ***'A-1'*** is the cumulative fraction of accesses with reuse distance up to '***d***' in the total of accesses.

Now for reuse distance, ***d***, which is equal to '***A***' and above, the problem is how to determine the hit rate given that the reuse distance obtained is not set reuse distance but for the entire cache memory. Due to the fact that we don't have access to set reuse distance, it is right to estimate the probability that for a given cache set ***X,*** which contain memory block ***E,*** that the reuse of memory block ***E***, is a hit. To be able to estimate that, certain assumptions have to be made to make this study tractable. The following assumptions were made: The study assumed that memory accesses are mapped into various caches line randomly; Mapping of cache lines into various caches sets are assume to be random; and Reuse distance distributions are identically and independently distributed (iid). There is need to define certain concept that will help in making the derivation of this cache model less cumbersome. If during the entire run of a program that processor reuses some memory blocks with reuse distance ***d***, the fraction of memory accesses with reuse distance ***d,*** will be the fraction memory accesses with ***d,*** to the total memory accesses. Now, reuse distance distribution R_d, is defined as the fraction of memory accesses with exactly reuse distance ***d,*** is given by (1).

$$R_d = \frac{Frequency\ of\ reuse\ distance, fd}{Total\ memory\ accesses, T} \quad (1)$$

where f_d=frequency of reuse distance equal to exactly ***d,*** and ***T***=total no. of memory accesses. Cumulative Reuse distance $R_{(i\leq d)}$ is also defined as the probability of obtaining at most reuse distance ***d,*** in the entire memory accesses, as in (2):

$$R_{(i\leq d)} = \sum_{i=0}^{d} R_d \tag{2}$$

In full associative cache, (2) gives the hit rate of the cache when ***d*** is equal to ***B-1*** where ***B*** is the number of the block in a cache. In set associative cache, (2) is used to determine the hit rate at reuse distance ***d***, that is less or equal to ***A-1*** where ***A*** is the associativity of the cache. In this study, the aim is to model set associative L_1 cache of which some of the reuse distance will be equal or more than the associativity of the cache.

Let's start by finding the probability of cache access being in cache set ***X***, the probability of an access being in set ***X*** is X_1=***1/S*** and the probability of not being in set ***X*** is X_0=***(***1-*1/S*)*.* So, assuming that X_1 is probability of success ρ and probability of failures X_0 is ***q***=(1- *1/S*). This can be stated in form of binomial probability by finding the probability of obtaining a success after ***d*** (reuse distance) using Bernoulli trials as shown in (3):

$$\mathrm{P}_{(\mathrm{k=Success})} = \sum_{k=0}^{d} \binom{n}{k} p^k q^{n-k} \tag{3}$$

where k=1, 2, 3…d, (**k** is number of success)

In memory block, reuses of cache line which the reuse distance is more than the associativity of the cache, the reuse can be cache hit only and only if the number of intervening accesses in cache set ***X*** is at most equal to ***A-1***. To calculate the probability of hit in reuse distance that is greater than or equal to the associativity ***A***, the probability of obtaining at most ***A-1*** unique intervening accesses in cache set ***X*** is obtained. This can be derived in form of cumulative binomial probability distribution function as shown in (4).

$$\mathrm{P}_{(\mathrm{k}\leq \mathrm{A}-1)} = \sum_{k=0}^{A-1} \binom{d}{k} p^k q^{d-k} \tag{4}$$

Substituting for ***ρ*** and ***q*** in (4) gives (5):

$$\mathrm{P}_{(\mathrm{k}\leq \mathrm{A}-1)} = \sum_{k=0}^{A-1} \binom{d}{k} \left(\frac{1}{S}\right)^k \left(\frac{\mathrm{S}-1}{\mathrm{S}}\right)^{d-k} \tag{5}$$

In (5) calculate the probability of hit in accesses with reuse distance equal or greater than ***A***. To calculate the hit rate in such reuse distance, we multiply the (5) with (1) the fraction of accesses that has that reuse distance which gives rise to (6) and (7).

$$\mathrm{P}_{(\mathrm{d_i})} = \frac{No.of\ accesses\ with\ reuse\ distance=d}{Total\ no.of\ memory\ accesses} * \sum_{k=0}^{A-1} \binom{d}{k} \left(\frac{1}{S}\right)^k \left(\frac{\mathrm{S}-1}{\mathrm{S}}\right)^{d-k} \tag{6}$$

$$\mathrm{P}_{(\mathrm{d_i})} = R_d * \sum_{k=0}^{A-1} \binom{d}{k} \left(\frac{1}{S}\right)^k \left(\frac{\mathrm{S}-1}{\mathrm{S}}\right)^{d-k} \tag{7}$$

In (6) and (7) gives the probability of hit in a given reuse distance equal or greater than degree of associative in cache. Then the total hit rate for all the reuse distance equal to ***A*** and above is given in (8).

$$\mathrm{P}_{(\mathrm{d}\geq \mathrm{A}_{\mathrm{Total}})} = \sum_{A}^{\infty} R_{d\,d} * \sum_{k=0}^{A-1} \binom{d}{k} \left(\frac{1}{S}\right)^k \left(\frac{\mathrm{S}-1}{\mathrm{S}}\right)^{d-k} \tag{8}$$

where R_d is reuse distance distribution of memory accesses with reuse distance=***d***.

In (8) gives cache hit rate for a memory block reuse with reuse distance greater or equal to associativity of a cache ***A***. In other to obtain the overall hit rate of an application giving an LRU cache with associativity ***A*** and number of set ***S***, addition of all the hit rate of every memory reuse distance distribution

including those with reuse less than or equal to ***A-1*** is required. Therefore, cache hit rate of an application is given as shown in (9) which is derived by obtaining the cumulative reuse distance distributions up to ***A-1*** and adding it to (8).

$$Hit\ Rate = \sum_0^{A-1} R_d + \sum_A^{\infty} R_d * \sum_{k=0}^{A-1} \binom{d}{k} \left(\frac{1}{S}\right)^k \left(\frac{S-1}{S}\right)^{d-k} \quad (9)$$

In (9) gives us the cache hit rate of an application in set associative cache with least recently used (LRU) replacement policy.

3. RESEARCH DESIGN

This section presents the data, tools, benchmarks and experimental setup for this study. We also explore the methods employed in the implementation of this study.

3.1. Metrics

In this experiment, we choose a microarchitecture-independent metric which characterized the memory usage of embedded system workloads. The metric been selected is reuse distance which was profiled using MICA Pintool [28]. This metric is microarchitecture independent because it only characterizes the memory behavior of workloads given a particular instruction set architecture (ISA). It is flexible to use and remain the same across different microarchitecture within the same ISA. Also, cache parameters such as cache sizes and configurations were also used along with this reuse distance to predict cache performance which in this case was cache hit rate.

3.2. Benchmarks

We used Mibench benchmarks [29] in the evaluation of the cache model presented. Mibench benchmarks are benchmark suite which follows EEMBC benchmarks [30] model. It is divided into six groups to represents the six domains of embedded systems. Three benchmarks were chosen from Mibench benchmark, two from automotive/industrial domain which are bitcounts and basicmath and FFT benchmark from network domain. These benchmarks were used to evaluate the model presented in this paper.

3.3. Tools

In order to profile the memory reuse distance, Intel Pin [23] a dynamic instrumentation engine was used along with Pintool called MICA developed by Eeckhout and Hoste [28]. This pin tool is capable of characterizing the memory reuse distance of embedded systems workload. The results of this tool are generated in bin with each bin representing the frequency of a particular range of reuse distance. MICA Pintool can be configured to profile a particular number of instructions or full instructions using its configuration file. The cache model presented is built using Bournoli cumulative binomial probability as shown in section 2. After evaluation of the model using the profiled reuse distance metrics and cache configuration parameters, we use sim-cheetah [31] from Simplescalar simulator suites [32] to simulate these selected benchmarks given the corresponding cache sizes and configurations.

3.4. Implementation

It is also important to note that all results from this study were generated using Intel® core (TM) i3 processor, Ubuntu 10.10 and gcc-3.4. Intel Pin-3.4 was used along with MICA_v0.40 Pintool to characterize the applications to generate reuse distance profiles. Sim-cheetah simulator from Simplescalar simulator suite v4.0 was used to compare the results of the cache model. We implemented the cache model using the following cache configurations; 2-ways, 4-ways, 8-ways and 16-ways and the cache sizes examined was 4kb, 8kb, 16kb, 32kb, and 64kb. The size of cache line used in both evaluation and sim-cheetah simulation was 32kb. These cache parameters were chosen to represent real world level one ***(L_1)*** data cache in an embedded processor. Three benchmarks were chosen from Mibench benchmark, two from automotive/industrial domain which are bitcounts and basicmath and FFT benchmark from network domain. These benchmarks were used to evaluate the model presented in this paper. For the model evaluation, these benchmarks were compiled using gcc-3.4 with O3 optimization. First, the selected applications after compilation were characterized into their memory reuse distance using MICA Pintool. Then we built a reuse distance histogram for the three benchmarks selected to show how characterized were the benchmarks, given their memory reuse distance. In order to evaluate the model, these memories reused distance (***d***) profiled along with corresponding cache parameters were substituted in our cache model to predict or to estimate the cache hit rate for given

application. For each benchmark, these steps were taking for the whole cache sizes and configurations, one at a time to arrive at the results presented. Finally, sim-cheetah was used to simulate these selected benchmarks by configuring it for the same cache sizes and configurations as in the cache model evaluation. The results of sim-cheetah give the cache miss rate which was converted to hit rate using the relationship between the two parameters. Both results from cache model and sim-cheetah were presented alongside each other in a table for comparisons. We also calculated the absolute mean errors between the cache model results and that from sim-cheetah simulator. These calculated errors were presented alongside the tables of results.

4. RESULTS AND DISCUSSION

In this section, we evaluate the model given in (9). But before that, we characterized the memory reuse distance of the three selected Mibench benchmarks and built their reuse distance histogram as shown in Figures 1-3. Figure 1 shows the reuse distance histogram of bitcount benchmark with more than 99.9 percent of the reuse distance having a reuse distance below 24 while the largest reuse distance for the benchmarks is 384. In Figure 2, which is the reuse distance histogram of Basicmath benchmark, the highest memory reuse distance obtained is 196608 as compared to 384 from bitcount. Also, more than 99.9 percent of the memory access in basicmath benchmark has its reuse distance at maximum of 192. Figure 3 shows the reuse distance histogram of FFT benchmark. It shows that its largest reuse distance is 49152 with frequency of 76562. Over 99.8% of the reuse distance has reuse distance below 384 meaning that the benchmark locality is good.

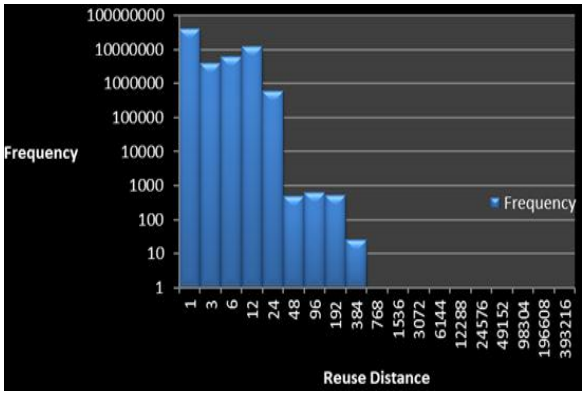

Figure 1. Reused distance histogram of bitcount benchmark

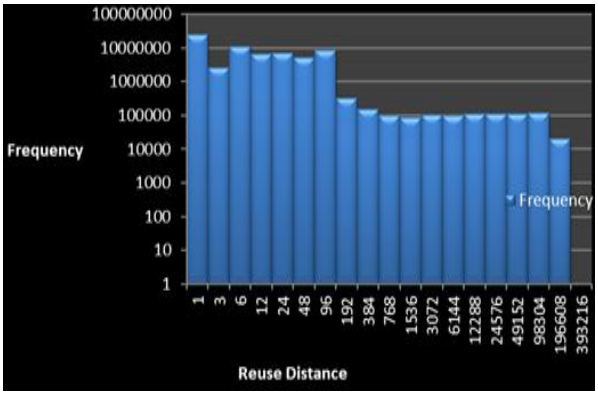

Figure 2. Reused distance histogram of basicmath benchmark

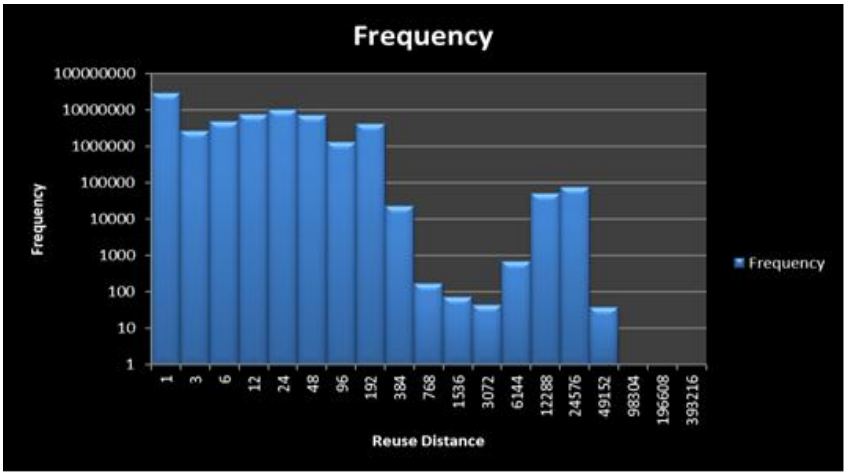

Figure 3. Reuse distance histogram of FFT benchmark

After characterizing the benchmarks, the reuse distance profiles were used to evaluate our cache model. Also, the benchmarks were simulated using sim-cheetah simulator under the same cache parameters as in cache model evaluation for comparison of results. Both results were shown alongside each other in a

table with the calculated absolute error between them. Tables 1, 2 and 3 shows the evaluated results obtained by using the cache model to estimate hit rate and that from sim-cheetah simulator for bitcount, basicmath and FFT benchmarks respectively for comparisons.

Table 1. Results on FFT benchmark for different cache configurations and sizes with errors

Cache Configurations	Percentage Hit Rates Using Proposed Model	Percentage Hit Rates Using Sim-cheetah	Absolute Errors
4kb 16-Ways	99.9986	99.9998	0.0012
4kb 8-Ways	99.9984	99.9998	0.0014
4kb 4-Ways	99.9891	99.9998	0.0107
4kb 2-Ways	99.6254	99.9998	0.3730
8kb 16-Ways	99.9002	99.9998	0.0996
8kb 8-Ways	99.9982	99.9998	0.0016
8kb 4-Ways	99.9993	99.9998	0.0005
8kb 2-Ways	99.9994	99.9998	0.0004
16kb 16-Ways	99.9995	99.9998	0.0003
16kb 8-Ways	99.9995	99.9998	0.0003
16kb 4-Ways	99.9994	99.9998	0.0004
16kb 2-Ways	99.9739	99.9998	0.0259
32kb 16-Ways	99.9930	99.9998	0.0068
32kb 8-Ways	99.9995	99.9998	0.0003
32kb 4-Ways	99.9995	99.9998	0.0003
32kb 2-Ways	99.9995	99.9998	0.0003
64kb 16-Ways	99.9995	99.9998	0.0003
64kb 8-Ways	99.9995	99.9998	0.0003
64kb 4-Ways	99.9995	99.9998	0.0003
64kb 2-Ways	99.9979	99.9998	0.0019
Mean Values	99.9734	99.9998	0.02629

Table 2. Results on basicmath benchmark for different cache configurations and sizes with errors

Cache Configurations	Percentage Hit Rates Using Proposed Model	Percentage Hit Rates Using Sim-cheetah	Absolute Errors
4kb 16-Ways	92.3929	99.8441	7.4512
4kb 8-Ways	91.7408	99.8357	8.0949
4kb 4-Ways	90.9682	99.8382	8.8700
4kb 2-Ways	89.3096	99.7936	10.4840
8kb 16-Ways	98.9793	99.9788	0.9995
8kb 8-Ways	98.2066	99.9751	1.7685
8kb 4-Ways	97.0596	99.9687	2.9091
8kb 2-Ways	95.1039	99.9263	4.8224
16kb 16-Ways	99.9504	99.9999	0.0495
16kb 8-Ways	99.8778	99.9999	0.1221
16kb 4-Ways	99.4657	99.9995	0.5338
16kb 2-Ways	98.2240	99.9907	1.7667
32kb 16-Ways	99.9546	99.9999	0.0453
32kb 8-Ways	99.9635	99.9999	0.0364
32kb 4-Ways	99.9023	99.9999	0.0976
32kb 2-Ways	99.4299	99.9999	0.5700
64kb 16-Ways	99.9546	99.9999	0.0453
64kb 8-Ways	99.9546	99.9999	0.0453
64kb 4-Ways	99.9503	99.9999	0.0496
64kb 2-Ways	99.8096	99.9999	0.1903
Mean Values	97.5099	99.9575	2.4476

Figures 4-6 also shows the graphs of cache model predicted and sim-cheetah simulated hit rate against different cache configurations for bitcount, basicmath and FFT benchmark. The summary of the results shows that the model mean errors for the cache model for bitcount, basicmath, and FFT benchmarks are 0.0263%, 2.4476%, and 1.9000% respectively. Hence, the model mean error for the three benchmarks is equal to 1.4579%. These results are comparable to the results obtained in [20] and [22] which gave model mean errors for LRU as 1.9% and 2.17% respectively.

Table 3. Results on FFT benchmark for different cache configurations and sizes with errors

Cache Configurations	Percentage Hit Rates Using Proposed Model	Percentage Hit Rates Using Sim-cheetah	Absolute errors
4kb 16-Ways	93.4509	99.8111	6.3602
4kb 8-Ways	93.4834	99.8119	6.3285
4kb 4-Ways	93.0136	99.6824	6.6692
4kb 2-Ways	91.0497	98.3705	7.3208
8kb 16-Ways	98.8278	99.8277	0.9999
8kb 8-Ways	98.1577	99.8302	1.6725
8kb 4-Ways	97.3649	99.8054	2.4405
8kb 2-Ways	95.8140	98.9278	3.1138
16kb 16-Ways	99.7724	99.8307	0.0583
16kb 8-Ways	99.7016	99.8307	0.1291
16kb 4-Ways	99.3466	99.8236	0.4770
16kb 2-Ways	98.3499	99.8167	1.4668
32kb 16-Ways	99.7802	99.8308	0.0506
32kb 8-Ways	99.7788	99.8308	0.0520
32kb 4-Ways	99.7321	99.8308	0.0987
32kb 2-Ways	99.3428	99.8297	0.4869
64kb 16-Ways	99.7802	99.8308	0.0506
64kb 8-Ways	99.7802	99.8309	0.0507
64kb 4-Ways	99.7761	99.8310	0.0549
64kb 2-Ways	99.6587	99.8310	0.1723
Mean Values	97.7981	99.7007	1.90000

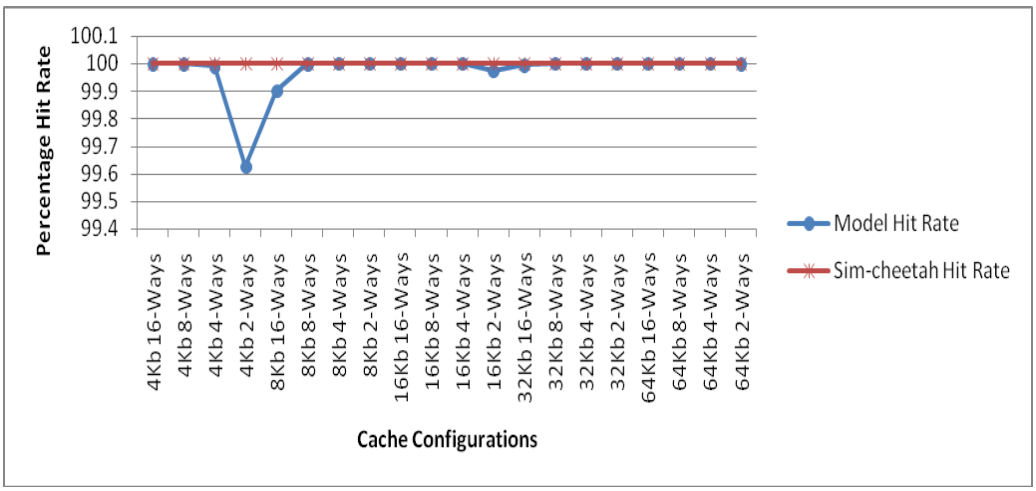

Figure 4. Comparison of model and sim-cheetah hit rate for bitcount benchmarks

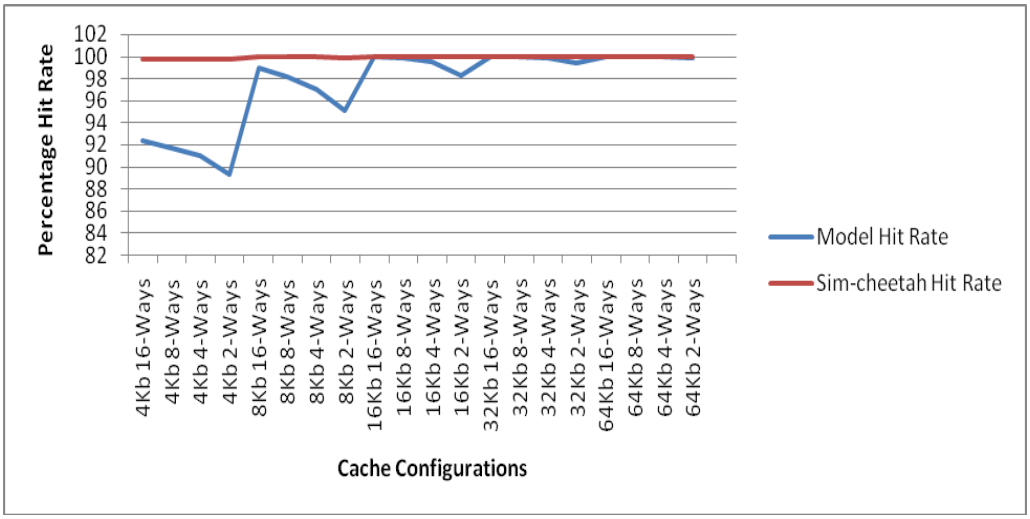

Figure 5. Comparison of model and sim-cheetah hit rate for basicmath benchmarks

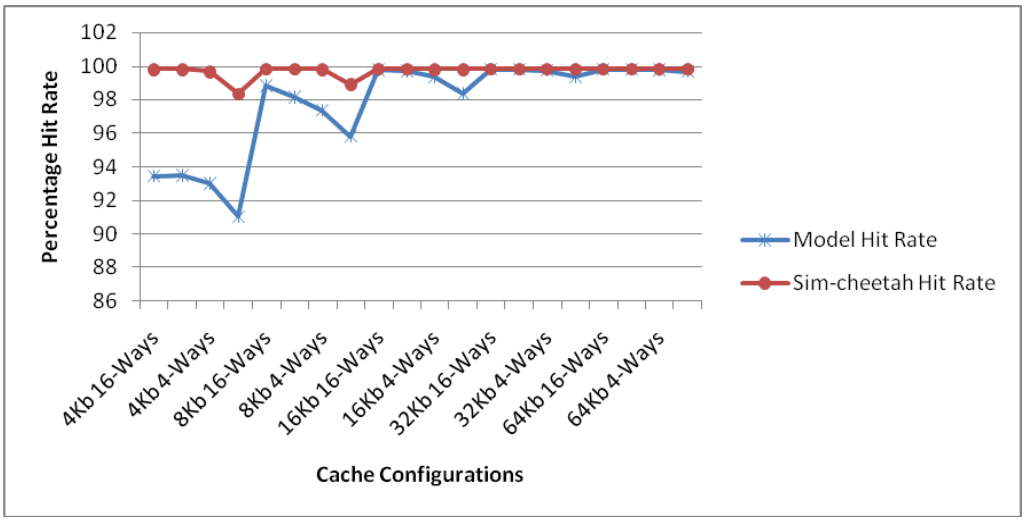

Figure 6. Comparison of model and sim-cheetah hit rate for FFT benchmarks

5. CONCLUSION

The embedded workloads were characterized to collect reuse distance metrics using MICA Pintool. This metrics with corresponding cache configuration parameters were applied to the developed model to estimate cache hit rate as shown in the results from this study. The results were compared using sim-cheetah from Simplescalar simulators suite. The margin of errors in results was below 5% and within the acceptable limits showing that the model can be used to estimate hit rates of cache and to explore cache design options. This model proved to be feasible since the results were comparable to other existing cache models. The results shows that the smaller the reuse distance, the better the performance of the cache. The results of this model were compared to simulated hit rate from standard simulator called sim-cheetah and it was observed to follows similar trend with an allowable margin of error. In conclusion, this model as presented can be used to estimate cache memory behavior with reasonable accuracy.

REFERENCES

[1] O. Mutlu and L. Subramanian, "Research Problems and Opportunities in Memory Systems," *Supercomputing Frontiers and Innovations: an International Journal*, vol. 1, no. 3, pp 19-55, 2014, doi: 10.14529/jsfi140302.

[2] M. Ferdman *et al.*, "Clearing the Clouds: A Study of Emerging Scale-Out Work-loads on Modern Hardware,". *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems*, March 2012, pp. 37-48, doi: 10.1145/2150976.2150982.

[3] M. V. Wilkes, "The Memory Gap and the Future of High Performance Memories," *ACM SIGARCH Computer Architecture News*, vol. 29, no. 1, March 2001, pp 2-7, doi: 10.1145/373574.373576.

[4] G. E. Moore, "Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff.," in *IEEE Solid-State Circuits Society Newsletter*, vol. 11, no. 3, pp. 33-35, Sept. 2006, doi: 10.1109/N-SSC.2006.4785860.

[5] Y. Kim *et al.*, "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors," *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, 2014, pp. 361-372, doi: 10.1109/ISCA.2014.6853210.

[6] V. Cuppu, B. Jacob, B. Davis and T. Mudge, "A performance comparison of contemporary DRAM architectures," *Proceedings of the 26th International Symposium on Computer Architecture (Cat. No.99CB36367)*, 1999, pp. 222-233, doi: 10.1109/ISCA.1999.765953.

[7] ARM Limited, "ARM1136JF-S and ARM1136J-S Technical Reference Manual," Manual ARM Limited, 2006.

[8] R. L. Mattson, J. Gecsei, D. R. Slutz and I. L. Traiger, "Evaluation techniques for storage hierarchies," in *IBM Systems Journal*, vol. 9, no. 2, pp. 78-117, 1970, doi: 10.1147/sj.92.0078.

[9] K. Beyls and E. H. D'Hollander, "Reuse Distance as a Metric for Cache Behavior," *in Proceedings of the Conference on Parallel and Distributed Computing and Systems*, 2001, pp. 617-662.

[10] Chi-Kang Chen, Hsin-I Wu, Cheng-Lin Tsai, "A Reuse-Distance Based Approach for Early-Stage Multi-Level Cache Design Optimization," in *Proceedings of Workshop on Synthesis And System Integration of Mixed Information technologies (SASIMI)*, 2018.

[11] Xiaoya Xiang, Chen Ding, Hao Luo, and Bin Bao, "HOTL: A higher order theory of locality," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 1, pp. 343-356, March 2013, doi: 10.1145/2490301.2451153.

[12] D. Eklov and E. Hagersten, "StatStack: Efficient modeling of LRU caches," *2010 IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS)*, 2010, pp. 55-65, doi: 10.1109/ISPASS.2010.5452069.
[13] Zhigang Hu, S. Kaxiras, and M. Martonosi, "Timekeeping in the memory system: Predicting and optimizing memory behavior," *ACM SIGARCH Computer Architecture News*, vol. 30, no. 2, pp 209-220, May 2002, doi: 10.1145/545214.545239.
[14] D. Eklov and E. Hagersten, "StatStack: Efficient modeling of LRU caches," *2010 IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS)*, 2010, pp. 55-65, doi: 10.1109/ISPASS.2010.5452069.
[15] S. T. Srinivasan, R. Dz-Ching Ju, A. R. Lebeck and C. Wilkerson, "Locality vs. criticality," *Proceedings 28th Annual International Symposium on Computer Architecture*, 2001, pp. 132-143, doi: 10.1109/ISCA.2001.937442.
[16] A. Jaleel, K. B. Theobald, S. C. Steely, and J. Emer, "High performance cache replacement using re-reference interval prediction (RRIP)," *ISCA '10: Proceedings of the 37th Annual International Symposium onComputer Architecture*, 2010, pp. 60-71, doi: 10.1145/1815961.1815971.
[17] S. Kumar and P. K. Singh, "An overview of modern cache memory and performance analysis of replacement policies," *2016 IEEE International Conference on Engineering and Technology (ICETECH)*, 2016, pp. 210-214, doi: 10.1109/ICETECH.2016.7569243.
[18] M. W. Ahmed &. M. A. Shah, "Cache Memory: An Analysis on Optimisation Techniques," *International Journal of Computer and Information Technology*, vol. 4, no. 2, pp. 414-418, 2015.
[19] Q. Wang, X. Liu and M. Chabbi, "Featherlight Reuse-Distance Measurement," *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019, pp. 440-453, doi: 10.1109/HPCA.2019.00056.
[20] X. Pan and B. Jonsson, "A modeling framework for reuse distance-based estimation of cache performance," *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2015, pp. 62-71, doi: 10.1109/ISPASS.2015.7095785.
[21] Yutao Zhong, Xipeng Shen, and Chen Ding, "Program locality analysis using reuse distance," *ACM Transactions on Programming Languages and Systems*, vol. 31, no. 6, pp. 245-257, 2009, doi: 10.1145/1552309.1552310.
[22] M. Brehob and R. Enbody, "An Analytical Model of Locality and Caching," Technical Report MSUCPS:TR99-31, Michigan State University, Department of Computer Science and Engineering, Michigan, 1999.
[23] Chi-Keung Luk *et al.*, "Pin: building customized program analysis tools with dynamic instrumentation," *ACM SIGPLAN Notices*, vol. 40, no. 6, June 2005, pp 190-200, doi: 10.1145/1064978.1065034
[24] J. L. Henning, "SPEC CPU2006 Benchmark Descriptions," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, September 2006, pp 1-17, doi: 10.1145/1186736.1186737
[25] N. Beckmann and D. Sanchez, "Modeling cache performance beyond LRU," *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2016, pp. 225-236, doi: 10.1109/HPCA.2016.7446067.
[26] T. Gysi, T. Grosser, L. Bradner and T. Hoefler "A Fast Analytical Model of Fully Associative Caches," in *PLDI 2019: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, June 2019, pp. 816-829, doi: 10.1145/3314221.3314606.
[27] P. J. Mucci, S. Browne, C. Deane, and G. Ho. "PAPI: A Portable Interface to Hardware Performance Counters," In *Proceedings of the Department of Defense HPCMP Users Groupconference*, 1999, pp. 7-10.
[28] K. Hoste and L. Eeckhout, "Microarchitecture-Independent Workload Characterization," in *IEEE Micro*, vol. 27, no. 3, pp. 63-72, May-June 2007, doi: 10.1109/MM.2007.56.
[29] 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*," Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538)*, 2001, pp. 3-14, doi: 10.1109/WWC.2001.990739.
[30] J. A. Poovey, T. M. Conte, M. Levy and S. Gal-On, "A Benchmark Characterization of the EEMBC Benchmark Suite," in *IEEE Micro*, vol. 29, no. 5, pp. 18-29, Sept.-Oct. 2009, doi: 10.1109/MM.2009.74.
[31] R. A. Sugumar and S. G. Abraham, Efficient simulation of multiple cache configurations using binomial trees. Technical Report CSE-TR-111-91, CSE Division, University of Michigan, 1991.
[32] T. Austin, E. Larson and D. Ernst, "SimpleScalar: an infrastructure for computer system modeling," in *Computer*, vol. 35, no. 2, pp. 59-67, Feb. 2002, doi: 10.1109/2.982917.

BIOGRAPHIES OF AUTHORS

V. C. Chijindu is a Senior Lecturer in the Department of Electronic Engineering, University of Nigeria, Nsukka, Nigeria. He obtained B.Eng and M.Eng in Electronic and Computer Engineering from Anambra State University of Technology Enugu Nigeria and Ph.D. in Computer and Control Engineering from Nnamdi Azikiwe University Awka Nigeria. His current research interests include artificial intelligence in medical diagnosis, digital systems design, signal/image processing, renewable energy systems and wireless sensor networks. E-mail: vincent.chijindu@unn.edu.ng

O. K. Ugwueze received his B.Eng in Electronics Engineering from University of Nigeria, Nsukka, Nigeria. He is currently a post graduate student in Computer and Digital Electronics at Electronic Engineering Department of the University of Nigeria, Nsukka. He is a member of Nigerian Society of Engineers. E-mail: kingsley.ugwueze@unn.edu.ng

C. C. Udeze received his B.Eng and M.Sc in Electronics and Computer Engineering from Nnamdi Azikwe University, Awka, Nigeria. He holds a PhD in Computer and Control Systems Engineering. He is a Senior Lecturer in Electronic Engineering Department of the University of Nigeria, Nsukka. He is a member of Nigerian Society of Engineers and has his COREN registration. E-mail: chidiebele.udeze@unn.edu.ng

M. A. Ahaneku obtained his B.Eng. Electrical/Electronic Engineering and MSc Communications Engineering from Federal University of Technology, Owerri, Nigeria in 1994 and 2000, respectively. He holds Ph.D in Communications Engineering from University of Nigeria, Nsukka. He is a Senior Lecturer in the Department of Electronic Engineering, University of Nigeria, and Nsukka. E-mail: mamilus.ahaneku@unn.edu.ng or ahamac2004@yahoo.co.uk

J. N. Eneh is currently a Senior Lecturer in the Department of Electronic Engineering of University of Nigeria Nsukka (UNN), She holds a Ph.D in Computer and Control Engineering Research Interest Areas Includes: Control System Networks, Robotics and Artificial Intelligence, Multi- Agent Systems, Optimal Control. E-mail: nnenna.eneh@unn.edu.ng

O. M. Ezeja holds a bachelor's degree in Electronic Engineering, a Master's and Doctorate degrees in Communication Engineering, all from the University of Nigeria, Nsukka, Nigeria. He is a Senior Lecturer in the Department of Electronic Engineering, University of Nigeria, Nsukka. His research interests include modelling handover algorithms for mobile communication networks, and wireless sensor networks (WSN). E-mail: obinna.ezeja@unn.edu.ng

E. C. Anoliefo received his B. A (Philosophy) in 1993 and B. Th (Theology) in 1998. He also holds a B.Eng., M. Eng. and Ph.D. in Electronic Engineering from University of Nigeria, Nsukka. He is a Senior Lecturer in the Department of Electronic Engineering, University of Nigeria Nsukka. E-mail: edward.anoliefo@unn.edu.ng