Advances in Computer Science Research, volume 80
International Conference on Computer Science, Electronics and Communication Engineering (CSECE 2018)

A Probability Model of Calculating L2 Cache Misses

Kecheng Ji[1], Ming Ling[1,*] and Li Liu[2]
[1]National ASIC System Engineering Technology Research Center, Southeast University, Nanjing, China
[2]Institute of Integrated Circuits Technology, Southeast University, Wuxi, China
[*]Corresponding author

***Abstract*—Stack or reuse distances have been widely adopted in studying memory localities and cache behaviors. However, the memory references, normally profiled by a binary instrumentation tool, only reflect the accessing sequence of instruction fetching and load or store executions. That is why the stack or the reuse distances obtained from these memory references cannot be used to predict the L2 or lower cache misses. This paper proposes a probability model to calculate the L2 reuse distance histogram from the L1 stack distance histograms without any extra simulations. The L2 cache misses or memory localities can be predicted fast and accurately based on the result of our model. We use 13 benchmarks chosen from *Mobybench* 2.0 and *SPEC 2006* to evaluate the accuracy of our model. With the support of *StatCache* and *StatStack*, the average absolute error of modeling the L2 cache misses is about 8%. Meanwhile, contrast to gem5 fast simulations, the process of predicting L2 cache misses can be sped up by 50 times on average.**

Keywords—probability model; stack distance; reuse distance; L2 cache misses

I. INTRODUCTION

For studying cache behaviors, the analytical models are normally fed with statistical memory characteristics, such as reuse or stack distance histograms [1]. However, these statistical characteristics are frequently obtained by profiling memory traces or emulating application executions [2], which only reflect memory behaviors in L1 caches. Therefore, lots of prior studies choose to use trace-driven simulations to predict L2 or lower level cache misses [3]. Although trace-driven simulations perform faster than detailed ones, the time overhead is still considerably larger than that of analytical models. Furthermore, the growing length of application traces and the non-unified simulator interfaces often bring storage and flexibility problems. Unfortunately, according to the study by [4], these problems have been more serious in recent years. On the other hand, the flexibility problem still remains as a big challenge and lots of frustrating coding as well as debugging works are needed. Last but not least, because they mainly focus on proposal evaluations rather than providing a guidance in architecture designs, simulation-based methods normally have rare architecture insights.

Therefore, this paper proposes a probability model to estimate the L2 reuse distance histogram from L1 stack distance histograms directly without any simulations. To our best knowledge, this model is the first analytical method to predict the L2 reuse distance histogram directly from the L1 stack distance histograms. The calculated L2 reuse distance histogram can be applied to predict L2 cache misses in single-core processors. Meanwhile, the study of contention behaviors in the multi-core shared cache can also take the advantage of fast modeling with the predicted results from our model.

The rest of the paper is organized as follows: Section II introduces how to calculate the L2 reuse distance while the model constructing is given in Section III. Section IV shows the experiment setup and studies the evaluation results. Finally, Section V concludes this paper.

II. CALCULATING THE L2 REUSE DISTANCE

A. Classical Stack Distance Theory

For studying cache behaviors, the stack distance is defined as the number of unique cache lines that accessed by the memory references during a reuse epoch, where a reuse epoch refers to the time interval between two successive memory references to the same cache line [5]. To collect the stack distance, prior studies usually construct the LRU stack history [6] that records latest references to different cache lines within each reuse epoch. In this way, the stack distance of current memory request can be easily calculated by counting the number of these latest references. Figure I shows a sequence of memory references to the L1 cache. X_i represents the i_{th} serviced reference while the accessed cache lines are labeled with the letters A, B, C and D. Although the cache line B has been accessed by 4 references X_4, X_7, X_9 and X_{10} within the reuse epoch of X_{11}, only X_{10} is recorded in the LRU stack history. Thus, the stack distance of X_{11} is 3 because there are 3 latest references X_5, X_8 and X_{10} in the reuse epoch of X_{11}. We define the accessed order of different cache lines within a reuse epoch as the LRU stack history of that epoch.

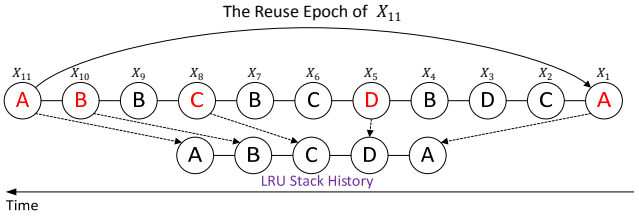

FIGURE I. LRU STACK HISTORY OF THE TWO MEMORY REFERENCES TO THE CACHE LINE A

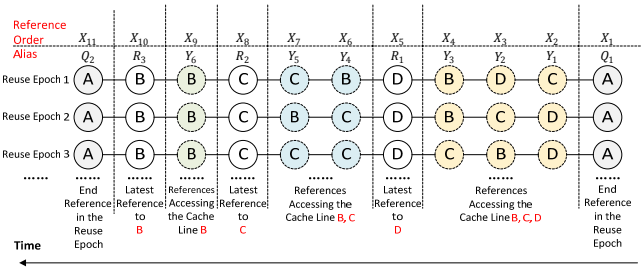

FIGURE II. SOME OTHER POSSIBLE REUSE EPOCH WITH THE SAME LRU STACK HISTORY IN FIGURE I

Normally, the collected stack distances of each reference are used to construct the so-called stack distance histogram for estimating LRU cache misses [5]. When the L1 cache has l cache lines, the number of cache misses can be calculated as (1) where $H_{s\_L1}(k)$ is defined as the L1 stack distance histogram.

$$Cache\ Misses = \sum_{k=l}^{\infty} H_{s\_L1}(k) \tag{1}$$

B. L2 Reuse Distance

Generally, memory references to the L2 cache are caused by L1 cache misses. Therefore, the L2 reuse distance of X_{11} in Figure I can be calculated by counting the number of L1 cache misses in the L1 reuse epoch of X_{11}. We define $P_{k\to r}$ to describe the ratio of how many L1 reuse epochs with stack distance k will generate r misses, or in other words, the L2 reuse distance r. In this case, the L2 reuse distance histogram $H_{r\_L2}(r)$ can be calculated by (2).

$$H_{r\_L2}(r) = \sum_{k=l}^{\infty} P_{k\to r} * H_{s\_L1}(k) \tag{2}$$

Typically, the references within each L1 reuse epoch can be classified into 2 groups. As shown in Figure I, the references of the first group, which is named as the set $\{NE\}$, do not generate reuse behaviors within the reuse epoch of X_{11}, such as X_2, X_3 and X_4. On the other hand, the references in the second group, which is named as the set $\{E\}$, have reuse epochs embedded in the X_{11} epoch, like X_5, X_6 etc. In this paper, we name these nested reuse epochs as "embedded epochs".

Across the whole program, there may be more than one possible reuse epochs that have the same LRU stack history. Figure II shows some other possible reuse epochs of X_{11}, which have the same LRU stack history shown in Figure I. For the given LRU stack history (B ← C ← D) within the epoch between X_1 and X_{11}, X_{10} must access the cache line B to guarantee the last accessed cache line is B. Meanwhile, according to the definition of LRU stack history, X_9 also must be a reference to the cache line B. In the same manner, the cache lines accessed by X_7 and X_6 are limited to B and C, while the references before X_5 can access anyone of the cache lines B, C and D. To classify above patterns, we give aliases for different references in the reuse epoch of X_{11}. The set $\{R_i\}$ represents the references that recorded as latest references in the LRU stack history. For example, X_5 has the alias of R_1 in Figure II. The end references in the target reuse epoch are denoted with the set $\{Q_i\}$. Lastly, other references in the epoch are named as the set $\{Y_i\}$.

1) L1 Cache Misses in the Set $\{NE\}$: Figure III simplifies the cases in Figure II to explain how to calculate $P_{k\to r}$ in this paper, where there is only one reference Y_1 and one embedded epoch that named as y'. Meanwhile, the L1 cache is always configured with 2 cache lines in the following discussions.

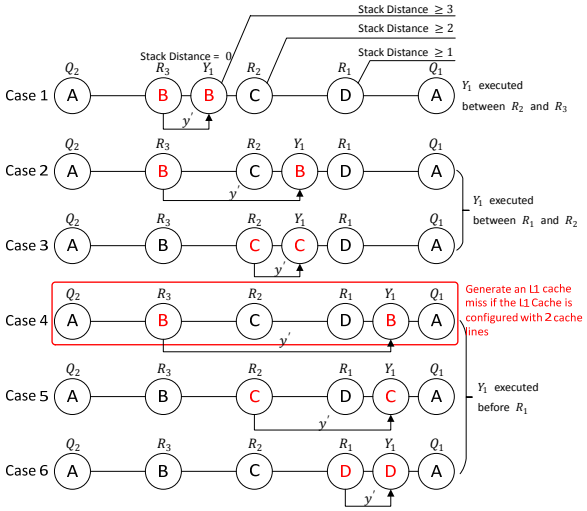

FIGURE III. ONLY ONE REFERENCE Y_1 IN THE REUSE EPOCH OF Q_2

As shown in the case 1 of Figure III, R_3 belongs to the set $\{E\}$ because it generates an embedded epoch y' with Y_1 (for the given LRU stack history shown in Figure I, the only possible cache line that accessed by Y_1 is B). The L1 stack distance of R_3 is 0. On the other hand, Y_1, R_1 and R_2 are the members of the set $\{NE\}$. The L1 stack distance of Y_1 is larger than or equal to 3 because 3 different cache lines (C, D, A) have been accessed before Y_1. Similarly, the L1 stack distances of R_1 and R_2 are larger than or equal to 1 and 2, respectively. There will be two possible cases, shown as case 2 and case 3, when Y_1 is serviced between R_1 and R_2 (Y_1 either accesses the cache line B or C). Meanwhile, there are 3 possible cases when Y_1 is executed before R_1, which are sketched as case 4 to case 6 (the cache line accessed by Y_1 can be anyone of B, C or D).

Although the references in the set $\{NE\}$ do not generate reuse epochs in the epoch of Q_2, their minimum L1 stack distances are determined, which have been analyzed before. For these references, the probabilities of generating L1 cache misses can be calculated using the L1 stack distance histogram. For each memory reference, we define P_m as the probability of generating an L1 cache miss given its L1 stack distance is larger than or equal to m. By generalizing the case of R_1, P_m can be calculated as (3) when the L1 cache has l cache lines. If l is smaller than or equal to m, the reference is certain to trigger an L1 cache miss. In this case, P_m should equal 1.

$$P_m = \begin{cases} \frac{\sum_{k=l}^{\infty} H_{s\_L1}(k)}{\sum_{k=m}^{\infty} H_{s\_L1}(k)}, & (l > m) \\ 1, & (l \le m) \end{cases} \tag{3}$$

To make the expression of $P_{k\to r}$ more concise and easier to calculate, we introduce one unified, weighted average probability P_{m_avg} to replace the usage of a group of different

P_m. According to (3), $\sum_{k=m}^{\infty} H_{s\_L1}(k) * P_m$ represents the number of L1 cache misses that generated by the references whose L1 stack distances are larger than or equal to m. By exploring all possible m, P_{m_avg} could be calculated as (4).

$$P_{m\_avg} = \frac{[\sum_{m=1}^{\infty} P_m * (\sum_{k=m}^{\infty} H_{s\_L1}(k))]}{\sum_{m=1}^{\infty} \sum_{k=m}^{\infty} H_{s\_L1}(k)} \quad (4)$$

2) L1 Cache Misses in the Set $\{E\}$: As shown in Figure III, there is only one case (case 4) that the embedded epoch y' generates an L1 cache miss (as we assumed before, the L1 cache has only 2 cache lines). In this paper, we assume that all possible cases of y' share the same probability to appear within the reuse epoch of Q_2. Therefore, the probability of y' to cause an L1 cache miss can be described as $1/6$ because there are 6 possible cases of y' in Figure III.

3) Generalization: P_y is defined as the probability of having an L1 cache misses in the set $\{E\}$. Based on the discussion in Figure III, P_y could be calculated as (5). The L1 cache has l cache lines while the L1 stack distance of Q_2 is k.

$$P_y \approx \frac{\sum_{m=0}^{k-l} \sum_{i=1}^{k-l-m} i}{\sum_{i=1}^{k} i}, \quad (k \geq l) \quad (5)$$

Actually, there may be more than one members in $\{E\}$. We observe that P_y could be calculated as (6) (represented as $P_y(n)$) approximatively while there are n references in the set $\{E\}$.

$$P_y \approx \frac{\sum_{m=0}^{k-l} \sum_{i=1}^{k-l-m} i}{\sum_{i=1}^{k} (i*n)}, \quad (k \geq l) \quad (6)$$

III. Model Generalization

A. The Probability of Multiple y'

As we mentioned above, there will be more y' epochs generated when more references Y_i are executed in the reuse epoch of Q_2. To describe the appearance ratios of different numbers of y', we use $P_e(k,n)$ to represent the probability of having n embedded epochs within the epoch of Q_2, given the L1 stack distance of Q_2 is k.

Given a reuse epoch q with stack distance s ($s \leq k$), the epoch q could be embedded in the reuse epoch Q_2 or not. Meanwhile, the epoch q can merely appear within the references whose L1 stack distances are larger than or equal to s, while the number of these references is $\sum_{a=s}^{\infty} H_{s\_L1}(a)$, in which $H_{s\_L1}(a)$ represents the L1 stack distance histogram. Thus, for Q_2 with L1 stack distance k, the probability of having one embedded epoch with the L1 stack distance s can be calculated as $H_{s\_L1}(k)/\sum_{a=s}^{\infty} H_{s\_L1}(a)$. Considering all possible s, $P_e(k,1)$ can be estimated as (7).

$$P_e(k,1) = \sum_{s=1}^{k-1} \left(\frac{H_{s\_L1}(k)}{\sum_{a=s}^{\infty} H_{s\_L1}(a)}\right) \quad (7)$$

Furthermore, assuming each embedded epoch is independent with each other, $P_e(k,n)$ can be calculated as (8).

$$P_e(k,n) = \left[\sum_{s=1}^{k-1} \left(\frac{H_{s\_L1}(k)}{\sum_{a=s}^{\infty} H_{s\_L1}(a)}\right)\right]^n \quad (8)$$

B. The Calculation of $P_{k\to r}$

Ultimately, the probability $P_{k\to r}$ can be obtained using (9) when the L1 stack distance equals k and the L2 reuse distance is r. For each reuse epoch, $P_e(k,n)$ represents the probability of having n reuse epochs y' (equals the members of set $\{E\}$). $C_n^x (P_y(n))^x * \left(1 - P_y(n)\right)^{n-x}$ denotes the probability of generating x L1 cache misses when there are n reuse epoch y' within the current epoch. For example, in Figure III, x is either 1 or 0 while n is 1 (because there is one and only one reuse epoch y' within the epoch of Q_2). Lastly, $C_k^{r-x} (P_{m\_avg})^{r-x} (1 - P_{m\_avg})^{k-r+x}$ gives the probability of causing $r - x$ L1 cache misses from the set $\{NE\}$.

$$P_{k\to r} = \sum_{n=1}^{\infty} [P_e(k,n) * C_n^x \left(P_y(n)\right)^x * \left(1 - P_y(n)\right)^{n-x} * C_k^{r-x} (P_{m\_avg})^{r-x} (1 - P_{m\_avg})^{k-r+x}] \quad (9)$$

IV. Evaluations

The cache architecture used for evaluations in this paper has two levels. From the CPU side to the memory side, the independent instruction and data caches are connected to the L2 shared cache through a crossbar. To simplify the discussion, all caches are fully-associative. However, it will be not difficult to extend our work to set-associative architectures. The replacement policy of L1 caches is set with the Least Recently Used (LRU) while that of the L2 cache could be configured as the Random or the LRU. The evaluating platform is implemented with gem5 *AtomicSimpleCPU* full-system simulator (containing two-level caches and DRAMs).

A. Evaluations of Predicting L2 Cache Misses

1) StatCache and L2 Cache Misses with the Random Replacement Policy: Figure IV shows the absolute errors of predicting L2 cache misses using the *StatCache* [7]. Briefly, the average absolute error of the tested benchmarks is around 8%. However, for some benchmarks, such as *BaiduMap* and *BBench*, the prediction errors increase significantly, up to 11.9%. The *StatCache* assumes that the memory references share the same L1 cache miss rate during a small execution slot. However, the correctness of this assumption depends on the length of profiling interval. The memory references of *BaiduMap* and *BBench* may not satisfy the assumption of *StatCache* when the profiling interval contains one million memory references in this paper.

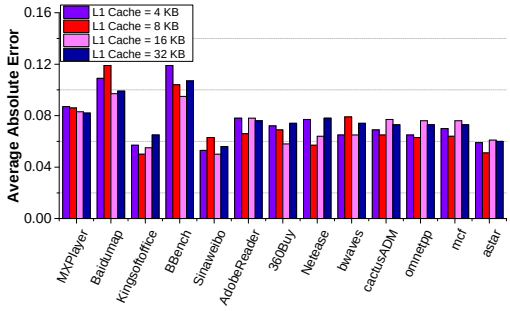

FIGURE IV. ABSOLUTE ERRORS OF PREDICTING L2 RANDOM CACHE MISSES

2) StatStack and L2 Cache Misses with the LRU Replacement Policy: StatStack [5] has provided an efficient method to calculate the stack distance histogram using the reuse distance histogram, which is accepted in this paper. Figure V shows the absolute errors of predicting L2 LRU cache misses. Most absolute errors are below 7%, which are lower than the errors in L2 Random cache miss predictions. This error reduction, we believe, could be caused by the error masking effects. For example, the reuse epoch, whose stack distance should be predicted as 8, is predicted with stack distance 9 using our model and *StatStack*. However, this reuse epoch will be regarded as a cache hit when the L2 cache is set with 16 cache lines, regardless the predicted stack distance is 8 or 9.

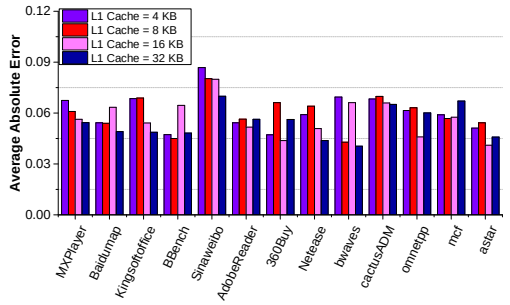

FIGURE V. ABSOLUTE ERRORS OF PREDICTING L2 LRU CACHE MISSES

B. *Time Overhead*

Figure VI shows the time overhead comparisons between gem5 *AtomicSimpleCPU* simulations and implementations of our model. The Y-axis gives the consumed minutes for predictions, which are shown in a log scalar. The X-axis represents all tested benchmarks. Briefly, the prediction can be sped by more than 50 times on average by using our model. Meanwhile, we can see that the time consumed in the L2 histogram calculation gives the half contribution to the total overhead of our model implementations approximately.

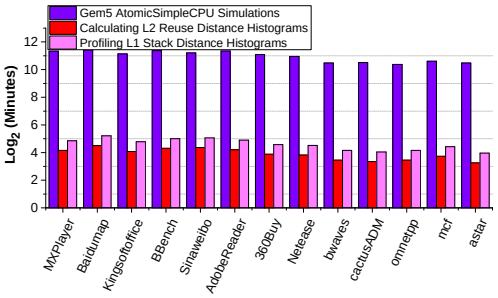

FIGURE VI. TIME OVERHEAD COMPARISONS BETWEEN GEM5 *ATOMICSIMPLECPU* SIMULATIONS AND OUR MODEL

V. CONCLUSION

This paper proposes a probability model to predict the L2 reuse distance histogram without any extra simulations. With the support of *StatCache* and *StatStack*, the calculated L2 histogram can be adopted for modeling L2 Random and LRU cache misses with the average errors of 8% and 6.8%, respectively. Our future works will extend the model application to multi-core architectures and consider the influences of the set-associative organization as well.

ACKNOWLEDGMENT

This work was supported under the Chinese National Mega Project of Science Research under Grant no.2014ZX01030101.

REFERENCES

[1] Tao Tang, Xuejun Yang, and Yisong Lin. Cache miss analysis for gpu programs based on stack distance profile. In Distributed Computing Systems (ICDCS), 2011 31st International Conference on, pages 623–634. IEEE, 2011.

[2] George Alm´asi, Clin Cas¸caval, and David A Padua. Calculating stack distances efficiently. In ACM Sigplan Notices, volume 38, pages 37–43. ACM, 2002.

[3] Richard A Uhlig and Trevor N Mudge. Trace-driven memory simulation: A survey. ACM Computing Surveys (CSUR), 29(2):128–170, 1997.

[4] Anastasiia Butko, Rafael Garibotti, Luciano Ost, Vianney Lapotre, Abdoulaye Gamatie, Gilles Sassatelli, and Chris Adeniyi-Jones. A tracedriven approach for fast and accurate simulation of manycore architectures. In Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific, pages 707–712. IEEE, 2015.

[5] David Eklov and Erik Hagersten. Statstack: Efficient modeling of lru caches. In Performance Analysis of Systems & Software (ISPASS), 2010 IEEE International Symposium on, pages 55–65. IEEE, 2010.

[6] Bryan T Bennett and Vincent J. Kruskal. Lru stack processing. IBM Journal of Research and Development, 19(4):353–357, 1975.

[7] Erik Berg and Erik Hagersten. Statcache: a probabilistic approach to efficient and accurate data locality analysis. In Performance Analysis of Systems and Software, 2004 IEEE International Symposium on-ISPASS, pages 20–27. IEEE, 2004.