ATLANTIS PRESS

Advances in Intelligent Systems Research (AISR), volume 155

2018 International Conference on Advanced Control, Automation and Artificial Intelligence (ACAAI 2018)

Quantifying the Influences of Data Prefetching Using Artificial Neural Networks

Kecheng Ji[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*—Data prefetching has been widely used in modern cache subsystems. Actually, an aggressive prefetching may bring negative yields unexpectedly, in which a new proposed prefetching strategy normally needs to be evaluated before being applied in the real design. In the last decade, prior researchers prefer to utilize the cycle-accurate simulations or trace-driven simulations to study the prefetching behaviors. However, as the increasing complexity of hardware components, the huge time-consuming simulation-based methods would never be appropriate for performance evaluations. This paper proposes a method of modeling prefetching influences on cache misses using artificial neural networks, which has an average error of 8% compared to gem5 cycle-accurate simulations, and the performance prediction process can be sped up by 30 times on average.**

Keywords—cache misses; data prefetching; artificial neural networks

I INTRODUCTION

As the computing power keeps increasing, most institutes select to use cycle-accurate or trace-driven simulations to evaluate their design proposals. However, due to the growing number of software and more hardware units integrated in the processors, the simulations-based methods have met the issues of frustrated code debugging and huge time-consuming. Furthermore, simulation-based methods can merely predict the performance on one benchmark on single hardware configuration, which cannot provide the trends between software characteristics and hardware configurable parameters.

Therefore, recent researchers start to focus on constructing analytical models, which have been an effective alternative to study the hardware performance. For study cache behaviors, previous works normally utilized the stack distance histogram to model the LRU-cache misses [1] or used the reuse distance histogram to analyze the Random-cache misses with a probability model [2]. However, both the stack and the reuse distance histogram need the support of profiling memory references within a binary instrumentation tool, in which these references actually reflects the cache behaviors without prefetching influences. Hence, this paper proposes a method based on the artificial neural networks (ANN) [3] to predict LRU-cache misses by fitting the relationship between the stack distance histogram and simulated cache misses with prefetching enabled. Meanwhile, our ANN model can predict cache misses under untrained benchmarks without any extra simulations nor ANN trainings, which can reduce the time overhead of model applications significantly.

II CLASSICAL STACK DISTANCE THEORY

Although the stack distance theory was used to model LRU fully-associative caches in early studies, it can be easily extended to set-associative caches by tracing memory requests in the granularity of each cache set [4]. Figure I shows a sequence of an address trace in one cache set, where memory references are labeled as X_i and accessed cache lines are represented with the letters A, B, C, etc. The arcs connect successive memory references to the same cache line to indicate a reuse epoch. The stack distance refers to the number of unique cache lines accessed during a reuse epoch. For example, considering the number of different memory requests between the two references to A, the stack distance of X_7 is 3. Since there are 3 unique accessed cache lines B, D and E between the two As.

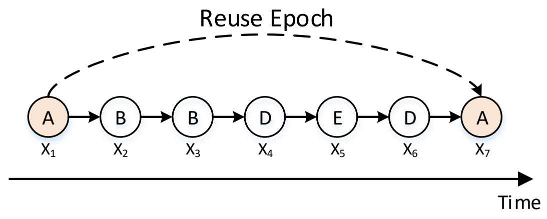

FIGURE I. REUSE EPOCH

If recording the stack distance of each memory request statistically, the histogram could be used to predict LRU-cache behaviors without prefetching directly. Specially, in a set-associative cache, the stack distance histogram of a whole program is constructed by the sum of all sub-histograms obtained from memory requests that accessing different cache sets. For example, $H_1(k)$ represents the stack distance histogram obtained from memory requests that accessing the first cache-set. Obviously, $H_1(k)$ can merely predict cache misses occurring in the first cache set. To predict total cache misses of all cache sets, all sub-histograms need to be integrated together (labeled as $H(k)$). Given that $H(k)$ represents the number of memory requests with the stack distance k of all cache sets while the parameter $assoc$ denotes the cache association, the cache misses can be calculated as Eq. 1 and N_{cs} gives the total number of cache sets.

$$Cache\ Misses = \sum_{k=assoc}^{\infty} H(k) = \sum_{k=assoc}^{\infty} \sum_{i=1}^{N_{cs}} H_i(k) \quad (1)$$

However, the prefetching would bring the potentially accessing data to the upper cache to reduce the cache misses during the program execution, in which the prefetching implementation would change the stack distances of memory references. For example, X_i, X_j and X_k in Figure II are generated by data prefetching, in which the stack distance of X_7 is changed from 3 to 5. Meanwhile, the number of reuse epochs will be increased due to the occurrence of X_i and X_k.

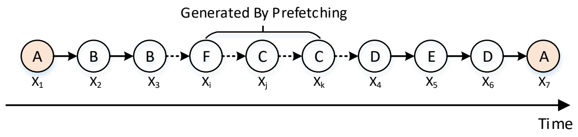

FIGURE II. PREFETCHING INFLUENCES ON PROFILED STACK DISTANCES

In summary, considering the effects of prefetching on the profiled stack distance histogram, some reuse epochs would migrate into nearby histogram bars while some reuse epochs in adjacent bars could move into the current bar. $H_r(k)$ represents the stack distance histogram considering prefetching influences, while $H_p(k)$ gives the one profiled from memory access traces in program order. $w_{l\to k}$ is the percentage of reuse epochs with stack distance changing from l to k ($l = k$ means the reuse epochs that are not migrated), while the $w_{k\to m}$ denotes the percentage of reuse epochs with stack distance changing from k to m ($m != k$). k denotes the stack distance of the current histogram bar. b_k represents the influences of new reuse epochs generated by prefetching. The first two terms in Eq. 2 could be described as $\sum_{i=1}^{\infty} w_i * H_p(i)$, which represents an effective result of reuse epochs migration from each bar of the $H_p(k)$ ($0 \le w_i \le 1$).

$$H_r(k) = \sum_{l=1}^{\infty} w_{l\to k} * H_p(l) - \sum_{m=1}^{\infty} w_{k\to m} * H_p(k) + b_k = \sum_{i=1}^{\infty} w_i * H_p(i) + b_k \qquad (2)$$

Combining Eq. 1 with Eq. 2, cache misses can be computed using Eq. 3 within the *assoc* way associative cache. Due to the extreme complexity and dynamics of stack distance changes by prefetching mechanisms, it is so difficult, if not impossible, to build a mechanistic model to quantify the set of $\{w_i\}$ and $\{b_k\}$. Therefore, we use the ANN to obtain the solution.

$$Cache\ Misses = \sum_{k=assoc}^{\infty} \left(\sum_{i=1}^{\infty} w_i * H_p(i) + b_k\right) \qquad (3)$$

III ANN MODELING

Figure III describes the ANN implementation flow in this paper. The training input is the profiled stack distance histogram while the training target is the number of cache misses obtained from gem5 cycle-accurate simulations with prefetching enabled. We collect the training materials by modifying the source code of gem5 simulator. To obtain multiple groups of training data, the training materials are dumped when every thread context switch occurs. Briefly, there are three major steps to construct the ANN model in this paper. First, we extract the stack distance histogram by profiling memory references. The corresponding cache miss statistics for each thread slice will also be collected in this step. Second, the optimal ANN training parameters, such as neurons in the hidden layer, activation functions and training methods will be chosen through experiments. Actually, we went through all possible candidates to find out the optimal parameters. Ultimately, we evaluate the constructed ANNs in aspects of self-predictions and cross-validations.

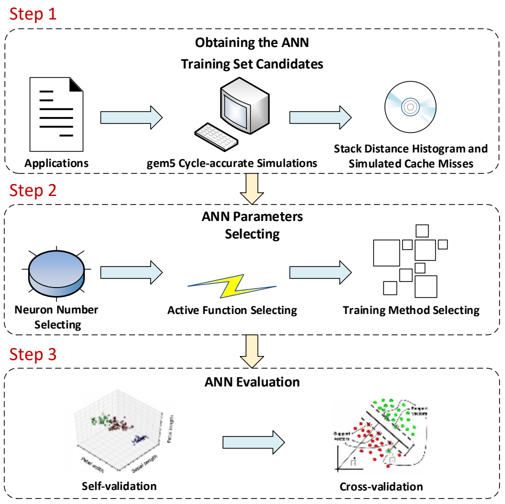

FIGURE III. ANN IMPLEMENTATION FLOW

For each cache set, we construct a single-linked list to record the LRU history of the memory references serviced in the current set. There are three basic steps to calculate the stack distance with these LRU histories. First, for a given memory reference, the address is masked to index the cache set. Generally, the address can be divided into three fields in the cache accessing, namely tag, set and block-offset. The field of set can be obtained through AND operations easily. Second, within the selected cache set, we can compare the tag address with all memory references by the cache line aligned address to determine whether the current memory reference causes a reuse epoch. If so, calculate the stack distance by counting the number of unique cache lines in the reuse epoch within the current cache set. For example, the reference X_i would cause a reuse epoch in Fig. 4, the number of unique cache lines between X_i and the last access to A is 1. Hence, the stack distance of X_i is 1. After the stack distance calculation, the single-linked list will be updated with the new coming memory reference at the tail of the list and the old one being erased from this list. Specially, if a new memory reference does not generate a reuse epoch, it will be inserted at the tail of the index list with an assigned stack distance of infinite, such as the reference X_j in Figure IV.

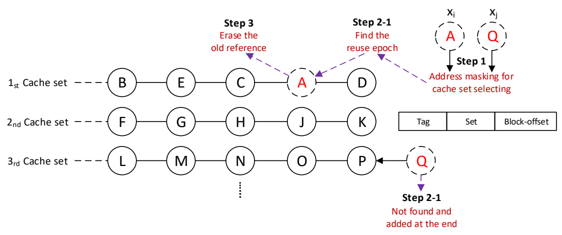

FIGURE IV. STACK DISTANCE CALCULATION

The ANN training needs selecting three parameters, such as the number of neurons in the hidden layer, the optimal activation function and the suitable training method. All these parameters are selected through detailed experiments. However, due to the page limit, the experiments results are not shown here but the detailed experimental steps could be found in our prior work [5]. Finally, the training parameters used in this paper are shown in Table I.

TABLE I. ANN TRAINING PARAMETERS

Neurons in the Hidden Layer	Training Method	Activation Function
16	trainlm	purline

IV EVALUATION

In this section, our ANNs will be evaluated with 10 benchmarks running on 4 hardware configurations (only L1 cache considered). The self-prediction error of the constructed ANNs represents the root mean square error of predicting cache misses under the trained benchmark. Figure V compares the prediction errors between using the classical stack distance theory and those of our ANN models. Obviously, our model increases the average prediction accuracy from 83% to 95% in the cases of "self-prediction".

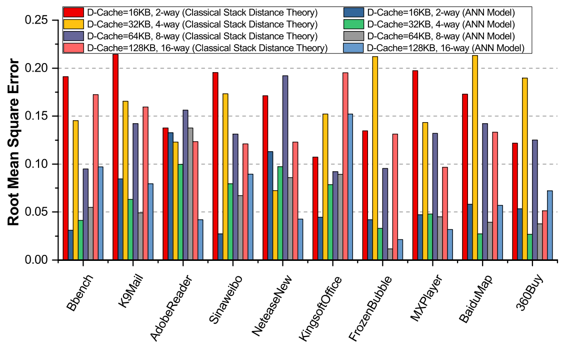

FIGURE V. ANN ACCURACY EVALUATION

Also, our model has the capability to predict cache misses on untrained workloads with a slight accuracy decreasing. Fig. 6 shows the prediction errors of 10 benchmarks we have tested while the hardware configurations are expressed under this figure. The labels at the right of Fig. 6 show the information of benchmarks for ANNs training. For example, the ANNs trained by K9Mail predict cache misses on BBench with the error being 0.095. As a result, the average root mean square error of benchmark cross-validation with untrained workloads may reach around 9%. Moreover, we also implemented benchmark cross-validations on other hardware configurations, which express similar accuracies as Figure VI. Due to page limitations, the results are not shown here.

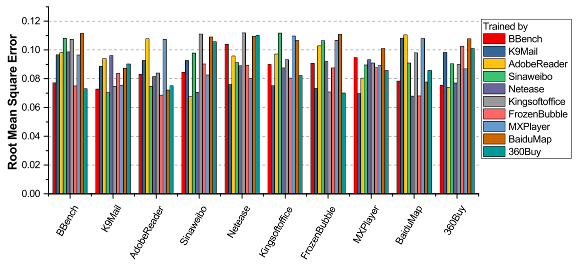

FIGURE VI. CROSS-VALIDATION ON DIFFERENT BENCHMARKS; D-CACHE = 32KB, 4-WAY

The time consumed in our model contains two parts. The first part is the time used in profiling the stack distance histogram, which can be evaluated using the gem5 fast simulation. The second part refers to time of obtaining simulated cache misses with prefetching enabled that should be measured using gem5 cycle-accurate simulation. To be precise, the time for ANN training is classified as the third part. Figure VII shows the time overhead comparison between gem5 cycle-accurate simulations and our ANN model. Due to the reusing of profiled stack distance histogram in different cache performance evaluations, our ANN model can speed up the prediction process by 30 times on average.

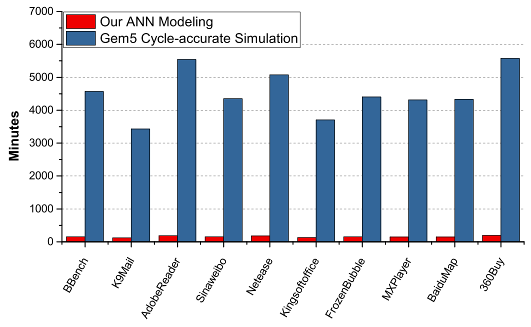

FIGURE VII. TIME OVERHEAD COMPARISON

ACKNOWLEDGEMENT

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

REFERENCES

[1] CaBcaval C, Padua D A. Estimating cache misses and locality using stack distances[C]//Proceedings of the 17th annual international conference on Supercomputing. ACM, 2003: 150-159.

[2] Berg E, Hagersten E. StatCache: a probabilistic approach to efficient and accurate data locality analysis[C]//Performance Analysis of Systems and Software, 2004 IEEE International Symposium on-ISPASS. IEEE, 2004: 20-27.

[3] Schalkoff R J. Artificial neural networks[M]. New York: McGraw-Hill, 1997.

[4] Almási G, Caşcaval C, Padua D A. Calculating stack distances efficiently[C]//ACM Sigplan Notices. ACM, 2002, 38(2 supplement): 37-43.

[5] Ji K, Ling M, Zhang Y, et al. An artificial neural network model of LRU-cache misses on out-of-order embedded processors [J]. Microprocessors and Microsystems, 2017, 50: 66-79.