Application Performance Prediction and Optimization Under Cache Allocation Technology

Yeseong Kim*, Ankit More**, Emily Shriver**, Tajana Rosing*
* *University of California San Diego* {yek048, tajana}@ucsd.edu
** *Intel Corporation* {ankit.more, emily.shriver}@intel.com

***Abstract*—Many applications running on high-performance computing systems share limited resources such as the last-level cache, often resulting in lower performance. Intel recently introduced a new control mechanism, called cache allocation technology (CAT), which controls the cache size used by each application. To intelligently utilize this technology for automated management, it is essential to accurately identify application performance behavior for different cache allocation scenarios. In this work, we show a novel approach which automatically builds a prediction model for application performance changes with CAT. We profile the workload characteristics based on Intel Top-down Microarchitecture Analysis Method (TMAM), and train the model using machine learning. The model predicts instructions per cycle (IPC) across available cache sizes allocated for the applications. We also design a dynamic cache management technique which utilizes the prediction model and intelligently partitions the cache resource to improve application throughput. We implemented and evaluated the proposed framework in Intel PMU profiling tool running on Xeon Platinum 8186 Skylake processor. In our evaluation, we show that the proposed model accurately predicts the IPC changes of applications with 4.7% error on average for different cache allocation scenarios. Our predictive online cache managements achieves improvements on application performance of up to 25% as compared to a prediction-agnostic policy.**

***Index Terms*—Performance prediction, cache allocation technology, top-down analysis methodology**

I. Introduction

Modern computing systems execute diverse applications that have significantly different requirements and priorities. The applications share limited resources such as the last-level cache (LLC) and memory. This consequently limits the quality of service (QoS) and performance of individual applications. For example, previous work showed that a few applications, also known as noisy neighbors, use disproportionately large amounts of shared resources in a cloud server [1].

To address the resource contention issue focusing on the on-chip cache, Intel recently announced a new control mechanism, called Cache Allocation Technology (CAT), as a feature of Resource Director Technology (RDT) [2]. The CAT mechanism has been supported in recent commercial processor families, e.g., Xeon E5 v4 processor. This technology allows allocating different amounts of cache sizes for the running applications. Prior research have also explored how to utilize the CAT technique for different optimization goals [3]–[7], e.g., providing system fairness and clustering cache ways for application groups. Some of the techniques estimate cache requirements of applications by utilizing historical traces of different metrics such as a set of performance monitoring counter (PMC) events. Although these management techniques are very important to effectively manage the limited cache resource in different scenarios, high-level models which accurately describe application performance behavior for different cache allocations are also essential for runtime managements.

This paper identifies a novel approach to the application performance prediction using Intel Top-down Microarchitecture Analysis Method (TMAM) [8]. The TMAM provides a robust way to quantify workload characteristics of applications. Since this method is readily available in most existing x86 microarchitectures, our approach can build the prediction models without the additional need for expert domain knowledge of target systems exploited in previous work [5]–[7].

In this paper, we show how the Intel TMAM metrics can describe application behavior changes for different cache allocations. We then present our modeling technique which automatically captures the relationship between the TMAM metrics and application performance changes with CAT. Our model estimates application performance in terms of instructions per cycle (IPC) across available cache sizes. We also present a model-based cache management technique for applications which have different characteristics and priorities. The management technique identifies the IPC changes by collecting the TMAM metrics, without ever having to run them on the new cache configurations. Based on the prediction results, it can intelligently partition the cache for each application.

We implement the proposed technique on Intel PMU profiling tool [9] running on Xeon Platinum 8186 Skylake processor. In our evaluation conducted with SPEC benchmark suites [10], the proposed model can accurately predict the IPC changes of applications running on CAT with only 4.7% average error. The experimental results also show that the model-based management policy improves application performance by up to 25% by utilizing our models for online optimization, as compared to another online management policy which does not use the prediction model.

II. Proposed Design

A. Overview of Proposed Design

Figure 1 shows the overview of the proposed framework, called Performance predictor and Optimizer for CAT, in short POCAT. POCAT is performed in two stages: an offline stage for prediction model building and an online stage for dynamic cache management based on the built model. In the online model building stage, it runs a set of benchmark applications, while collecting the TMAM metrics for different cache size settings. With the collected TMAM metrics, we build the prediction model with three steps: *automated modeling*, *feature selection*, and *model complexity optimization*. The result is the final prediction model, which uses TMAM metrics as the inputs and predicts IPC of an application over all available cache size settings.

Our prediction model can be exploited for various performance optimization and management problems. We propose a dynamic cache management technique that allocates cache

978-3-9819263-2-3/DATE19/©2019 EDAA

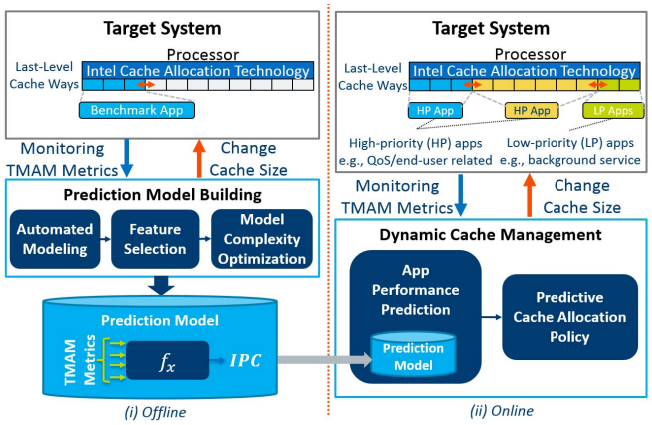


Fig. 1. Overview of Proposed POCAT Framework

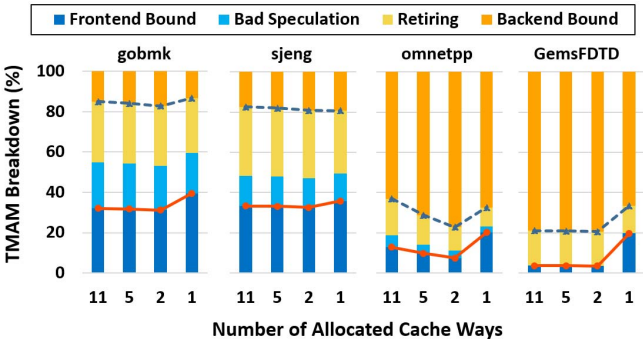


Fig. 2. TMAM Metric Changes with CAT

sizes for different applications to maximize average IPC on a target system in which each application has either high or low priority. POCAT monitors the TMAM metrics online to predict IPC of applications across available cache settings, and activates CAT with the identified optimal cache configuration.

B. Workload Analysis Based on TMAM

The proposed framework utilizes the TMAM metrics [8] to understand how the application behavior changes for different cache allocation configurations. Since the TMAM works on most x86 processors, POCAT can capture the application profiles without losing generality for various microarchitectures. It also simplifies the event selection procedure which is not often an obvious task for application performance analysis [11]. The TMAM metrics have four top-level categories: 'frontend bound', 'bad speculation', 'retiring', and 'backend bound'. At a high level, the 'frontend bound' and 'backend bound' metrics account for two major pipeline components of modern processors; the frontend fetches and decodes the program code, i.e., architectural instructions, whereas the backend monitors the availability of operands for the decoded micro operations to execute them. Each top-level metric has their subevents in a tree structure, e.g., 'L3 bound' is a descendant event of the 'backend bound' event; 'ICache misses' belongs to the 'frontend bound' event.

We present our analysis on experiments conducted on Xeon Platinum 8168 CPU at 2.70GHz with SPEC2006 suite. The processor supports the CAT mechanism for its LLC (L3) which has 11 ways in total. Figure 2 shows how the number of allocated cache ways influences on the TMAM metrics for four representative benchmarks. Since the 'frontend bound' metric for `gobmk` and `sjeng` benchmarks takes a relatively high percentage, we can classify them as compute-intensive workloads. In contrast, the `omnetpp` and `GemsFDTD` are memory-intensive benchmarks, as indicated by the high percentage value of the 'backend bound' metric.

In the analysis, we observe that the workload characteristics are changed with CAT primarily due to two factors: i) *LLC utilization* and ii) *instruction/data cache contention*.

LLC utilization: As shown in the results, the 'backend bound' metric typically decreases when reducing the number of allocated cache ways toward 2. This trend occurs for both the compute-intensive and memory-intensive workloads. The increased amount of the 'backend bound' metric varies for different benchmarks. For example, `omnetpp` shows higher increment in the 'backend bound' than `GemsFDTD`, although both are memory-intensive benchmarks. It is because the processor can handle most cache requests of `GemsFDTD` in upper-level caches such as L1 and L2. For example, a sub-level TMAM metric, 'L3 bound', belonging to the 'backend bound' is 28% for `omnetpp` and 8% for `GemsFDTD`.

Instruction/data cache contention: Limiting the cache size to a single way shows a distinct behavior in the TMAM metrics. In this case, the 'frontend bound' metric increases due to the growth of its sub-level metric, 'ICache misses', representing that the instruction and data caches compete for each other in the single way.

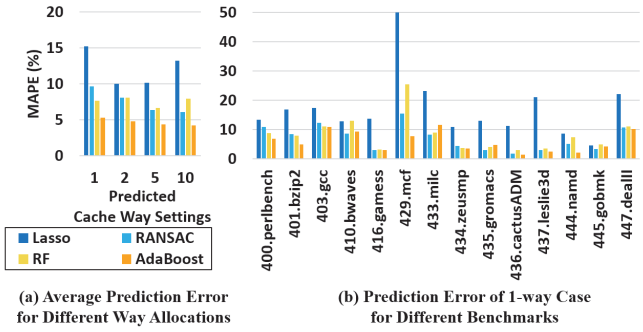
C. Prediction Model Building

POCAT automatically captures the relationship between the TMAM metrics and IPC behavior using machine learning. We create the machine learning dataset by collecting the TMAM metrics and IPC values of benchmark applications for each cache way setting. We denote $\mathbf{v}_x$ as the vector for all TMAM metrics (including sub-level metrics such as 'L3 bound') sampled when x ways are allocated for the benchmark execution. The *initial model* trained by POCAT is defined as follows:

$$f_x^y(\mathbf{v}_x) = IPC_{ratio} = \frac{IPC_y}{IPC_x}$$

where IPC_x is the IPC monitored with $\mathbf{v}_x$ for the x-way configuration and IPC_y is the IPC to predict for the y-way allocation. Since the model usually needs to be self-explanatory for in-depth managements and run efficiently for the next online prediction, we evaluated four light-weight white-box modeling approaches: (i) least absolute shrinkage and selection operator (Lasso) [12], which builds a linear regression model while selecting features with regularization, (ii) random sample consensus (RANSAC) [13] which handles outliers of the Lasso model, (iii) random forest (RF) [14] and (iv) AdaBoost.R2 [15]. The last two approaches utilize regressor decision trees as its base learner.

Figure 3 shows the comparison of cross-validated prediction accuracy for the different approaches. The TMAM metrics are measured for the 11-way case and the model predicts the IPC for the other cache configurations. For the error metric, we use mean absolute percentage error (MAPE). We observe that the tree-boosting AdaBoost model shows the best accuracy among the tested models. For example, as shown in Figure 3a, the tree



(a) Average Prediction Error for Different Way Allocations

(b) Prediction Error of 1-way Case for Different Benchmarks

Fig. 3. Comparison of Different Modeling Methodologies

boosting model outperforms Lasso model by 7.5% on average. The underlying reason is that the IPC is usually changed when the L3 usage exceeds the maximum capacity currently allocated. The tree-boosting model is able to automatically capture the appropriate threshold with 'L3 Bound' metric. In addition, as shown in Figure 3b, the AdaBoost model also outperforms other methodologies for the 1-way case which is related to the instruction/data cache contention.

Once the initial model is trained, POCAT optimizes the model complexity with two steps, *feature selection* and *model complexity optimization*. In the feature selection step, POCAT retrains the model through an iterative process by using the recursive feature elimination method, while only considering the last-level events. In our experiment, it selects sixteen events and the model trained with the events has a negligible difference in the prediction accuracy of less than 0.1%. The model complexity optimization step creates a compact form of the model that predicts multiple cache settings by exploiting multivariate estimation of AdaBoost. Depending on the configuration, it may create another model further optimized, called *all-way* model with multivariate estimation, which predicts all pairs of the cache settings by taking the current cache setting as an additional input of the model.

D. Dynamic Cache Management

We utilize the application performance model to design our predictive dynamic cache management technique. In our optimization scenario, we assume that there are either high or low priority applications. In practice, the QoS/end-user related applications can be the high priority applications, while existing background services can be included in the group of the low priority applications (LP apps.)

Our approach monitors the selected TMAM events for HP apps running on the system, and predicts the IPCs of each app if it were executing on different cache way sizes. This creates a $H \times (W - t)$ matrix of IPC prediction results where H is the number of HP apps, W is the number of ways, and t ways are allocated at least for the LP apps. Our management technique identifies the optimal combination of cache ways for the HP apps so that it minimizes a given cost function. The cost function decides the detailed optimization goal based on the IPC prediction table. For example, if the goal is to provide the fairness of IPCs for similar HP apps, the standard deviation of the predicted IPCs would be a suitable metric to evaluate the costs. In our implementation, we use the IPC loss as the cost function to maximize the total IPC of all HP apps. This strategy can be formulated with the $find\_optimal$ function, which is solvable using dynamic programming:

$$
\begin{aligned}
&find\_optimal(\mathcal{A}, N \text{ ways}) \\
&= \underset{app \in \mathcal{A}, n \geq 1}{\text{argmin}} \big(find\_optimal(\mathcal{A} - \{app\}, N - n \text{ ways}) \\
&\quad + cost(app, n \text{ ways}) \big)
\end{aligned}
$$

where $\mathcal{A}$ is the set of HP apps and $cost(app, n \text{ ways})$ is the IPC difference for app if the allocation is changed to n ways.

This function returns the minimal loss (or maximum benefit) in terms of the total IPC for all HP apps. During the solution computation, we can also identify the optimal cache allocation setting for each application. For example, let us assume that an application is predicted to achieve a higher IPC if one more cache is allocated. In this case, it checks if there is an application whose IPC loss is less than the improvement of the other. We conservatively change the cache allocation settings when the expected IPC benefit is larger than a configurable threshold. We set the threshold to 5% which is the average error of the prediction model.

III. Experimental Results

A. Experimental Setup

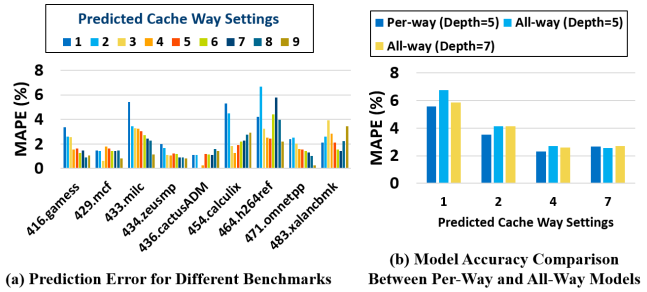
We have implemented the proposed framework by modifying Intel PMU profiling tool [9], and controlled the CAT features using MSRs [2]. For the statistical analysis, we exploited Scikit-learn library. The developed framework is evaluated on Xeon Platinum 8168 skylake CPU running at 2.70GHz, which has 48 physical cores. The LLC (L3) cache has 11 ways, in total 33 MBytes. We use the SPEC2006 for benchmark suite [10]. To evaluate the optimization policy, we generate random mixes of benchmarks. We vary the number of HP apps (H) and the number of LP apps (L), and evaluated with 100 different random mixes for each H and L combination. The number of threads of each benchmark application is also randomly set in the range from 1 to 4.

We collect the TMAM events at a rate of 2 seconds, which is the highest sampling rate of the baseline Intel PMU profiling tool. The runtime overhead for both the model-based prediction and online management technique is minimal, e.g., less than 50 ms for each sample in our experimental setup. The experimental results for the proposed prediction and optimization techniques are cross-validated using the leave-one-out method, i.e., evaluating each benchmark by separating the tested program from the training set.

B. Model Accuracy

Figure 4a shows the cross-validated accuracy of the *per-way* model for representative benchmarks, when the TMAM metrics are collected for the 10-way CAT setting. The result shows that the per-way model accurately predicts the IPC changes only with 4.7% error on average. Predicting the IPC of the 1-way case presents a relatively higher error, but even in this case, the worst case error is less than 7%. Figure 4b compares the per-way model to the all-way models. The result shows that, to achieve the similar level of accuracy, the all-way model requires more tree depths than the per-way model. For example, when the depth of the all-way model is 7, it shows the similar accuracy to the 5-depth per-way model.

We observe that the per-way model achieves the high accuracy in general. It is because the per-way model creates



(a) Prediction Error for Different Benchmarks

(b) Model Accuracy Comparison Between Per-Way and All-Way Models

Fig. 4. Prediction Accuracy of Proposed Models

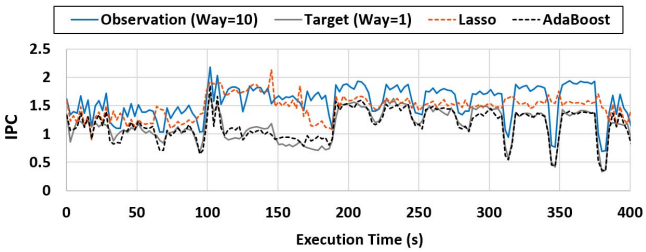


Fig. 5. Example of Prediction Results (`bzip2`, Per-way Model)

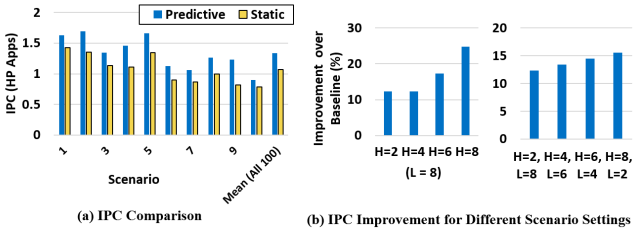
AdaBoost models for each of available cache ways, leading to the detailed prediction for each different case, whereas the per-way model only creates 1 model for the target system. However, the all-way model consumes 60% less memory than the per-way model.

Figure 5 shows an example of the prediction result for the `bzip2` benchmark, when the TMAM metrics are collected for the 10-way case and the models predict IPCs for the 1-way case. The result presents that the proposed tree-boosting model (AdaBoost) is better than conventional linear regression model (Lasso), in particular when there is a relatively large change in IPC due to the lack of the cache capacity. For example, the proposed model shows superior accuracy for the duration from 110 to 180 seconds.

C. Evaluation of Optimization Policy

Figure 6 shows the evaluation results of the dynamic cache allocation management. We exploit the per-way model for the IPC prediction, and compare our technique with a model-agnostic policy, called *static*, which allocates 1 way to LP apps and splits 10 ways best equally to each HP app. For example, when there are 4 HP apps, they will be allocated with 2, 2, 3, and 3 ways, respectively. This scenario is similar to the one that the work in [3] aimed to optimize. Figure 6a summarizes the results of the 100 random mixes when there are 8 HP apps ($H = 8$) and 8 LP apps ($L = 8$). The results show that that the predictive management technique outperforms the static policy by identifying the suitable cache size for each HP app based on the prediction models. For example, the predictive management of POCAT achieves 25% better IPC on average.

Figure 6b summarizes the performance improvement achieved for different H and L combinations. We observe that, when more HP apps exist, the predictive policy achieves better improvement, since the HP apps are more likely to compete with each other in the limited cache size, requiring the more elaborate cache size adjustments.



(a) IPC Comparison

(b) IPC Improvement for Different Scenario Settings

Fig. 6. Impact of Management Policy on IPC

IV. Conclusion

In this paper, we present a novel prediction model that automatically identifies the IPC behavior when using the Intel cache allocation technology. We analyze how the application TMAM behavior and performance are affected by cache sizes. Using the tree-boosting machine learning algorithm, the model predicts the future IPC changes with 4.7% error without changing the actual allocation. We show that the model can be exploited to build a predictive cache management policy. The experimental optimization policy achieves better application performance by up to 25% than the prediction-agnostic policy.

V. Acknowledgement

This work was supported by Intel Corporation, NSF grant #1730158 and #1527034.

References

[1] Intel. Quiet noisy neighbor with intel resource director technology. *White paper*, 2017.
[2] Intel. Introduction to cache allocation technology in the intel xeon processor e5 v4 family. *White paper*, 2016.
[3] Ioannis Papadakis, et al. Improving qos and utilisation in modern multi-core servers with dynamic cache partitioning. In *Proceedings of the Joined Workshops COSH 2017 and VisorHPC 2017*, 2017.
[4] Vicent Selfa, et al. Application clustering policies to address system fairness with intels cache allocation technology. In *Parallel Architectures and Compilation Techniques (PACT), 2017 26th International Conference on*, pages 194–205. IEEE, 2017.
[5] Lucia Pons, et al. Improving system turnaround time with intel cat by identifying llc critical applications. In *European Conference on Parallel Processing*, pages 603–615. Springer, 2018.
[6] Yaocheng Xiang, et al. Dcaps: dynamic cache allocation with partial sharing. In *Proceedings of the Thirteenth EuroSys Conference*, page 13. ACM, 2018.
[7] Cong Xu, et al. dcat: dynamic cache management for efficient, performance-sensitive infrastructure-as-a-service. In *Proceedings of the Thirteenth EuroSys Conference*, page 14. ACM, 2018.
[8] Ahmad Yasin. A top-down method for performance analysis and counters architecture. In *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*, pages 35–44. IEEE, 2014.
[9] Intel pmu profiling tools. https://github.com/andikleen/pmu-tools.
[10] Spec2006. https://www.spec.org/cpu2006/.
[11] Yeseong Kim, et al. P4: Phase-based power/performance prediction of heterogeneous systems via neural networks. In *Computer-Aided Design (ICCAD), 2017 IEEE/ACM International Conference on*, pages 683–690. IEEE, 2017.
[12] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.
[13] Martin A Fischler et al. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.
[14] Andy Liaw, et al. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.
[15] Harris Drucker. Improving regressors using boosting techniques. In *ICML*, volume 97, pages 107–115, 1997.