Facilitating Profile-Guided Compiler Optimization with Machine Learning

Yang Yang
College of Computer Science and Technology, Jilin University
Changchun, China
yangyang1519@mails.jlu.edu.cn

Xueying Wang
Institute of Computing Technology, Chinese Academy of Sciences
Beijing, China
wangxueying@ict.ac.cn

Guangli Li*
Institute of Computing Technology, Chinese Academy of Sciences
Beijing, China
liguangli@ict.ac.cn

Abstract

Profile-guided optimization (PGO) has been proven to be an effective compilation technique to improve program performance. However, classic PGO needs to collect numerous program runtime information through dynamic profiling for further compiler optimization, which leads to an expensive cost, hindering its applicability for large-scale applications.

In recent years, machine learning (ML) models have been introduced to guide compiler optimization, which can predict program information with a slight cost. While delivering promising performance, there are numerous adjustable configurations when training ML models and integrating them with compilers, including model structures, features, and predicted categories. As such, it is still challenging to design an effective ML-aided compiler optimization system.

In this paper, we use branch prediction as an example and perform several experimental studies on an AMD Ryzen CPU platform so as to explore the design of ML-aided PGO. We treat branch-taken frequency as probability, divide it into several categories and transform the prediction task into a classification problem by assigning pre-defined branch weights. We collect over 2,000,000 branches from widely-used benchmarks and utilize an instrumentation pass that is integrated with LLVM to subtract program features. We employ XGBoost, a commonly-used ML model, which can provide a good balance between accuracy and overhead, to predict the branch weights (i.e., probability). The programs will be optimized with the predicted branch weights, thereby avoiding the overhead of the profiling process.

We perform three experimental studies to analyze the design of ML-aided PGO as follows.

Analysis 1: The Number of Categories. As we treat our task as a classification problem, how to assign the category to each branch needs to be considered. The branch density indicates a strongly two-head distribution, which guides us to form a three-class classification task. We test a few kinds of partition policy based on intuition, our evaluation on a 2:8 partition performs the best accuracy, which reaches 93%.

Analysis 2: The Criteria of Classification. To further prove our intuition, we conduct an empirical analysis of the category partition rules. Both intuition-based division and equal division are tested, which proves that a three-way partition matches the distribution and performs better. And we find that in our tested programs, there are two different kinds of performance variation (improve or decrease) when the partition changed.

Analysis 3: The Number of Features. We evaluate our model's sensitivity to the features and its format. By constructing a matrix of feature pair's Pearson product-moment correlation coefficients, we can remove 16 features with only 1% accuracy loss. And we keep removing features iteratively to observe accuracy and speedup, and we find that the model's sensitivity to features is strongly program-dependent cause branch behavior differs from each other.

Results and Conclusion. We implement a prototype system of ML-aided PGO based on the above analysis, which employs predicted weights, rather than realistic profiling weights, for branch probability. Evaluation with representative real-world applications and *Polybench* benchmark suite demonstrates the effectiveness of our method, achieving an average of 1.03× and 1.95× speedups over the baseline (i.e., the programs without PGO), respectively. Moreover, the performance of our ML-aided PGO is very close to the classic PGO (1.05× and 1.97× speedups over the baseline) while reducing 58.3% and 94.8% optimization costs.

*Corresponding author.