Compilation Optimization Report.

Malachi Gage Sanderson.

Embry-Riddle Aeronautical University, CS332.

Fall 2021.

The topic of optimization is and has been one of the most pertinent areas of the field of computer science. Anyone working with a computer system or piece of software to any professional degree will work with trying to optimize a system at some point. One of the lowest level architectural forms of optimization is that of compiler optimization. An optimizing compiler is one that attempts to take some aspects of code such as its execution time, throughput, I/O and CPU execution overhead as well as many other performance-related attributes and minimize or maximize them to improve performance or efficiency. There are many different optimization algorithms and techniques such as loop unrolling and constant propagation, but this paper will discuss an optimization technique linked to Machine-Learning proposed by Li Fengqian (and his associates) in the 2014 research paper titled, *Feature Mining for Machine Learning Based Compilation Optimization*. In Li's paper he discusses a form of compilation optimization that utilizes machine-learning with an emphasis on "program features" and even implements a "feature extractor to extract feature values at specific phases and predict optimization plan dynamically" (Li, 2014). The resulting compilation optimization system that was implemented and tested produced results that had 5% increased average speed "for object file running speed than GCC O3 optimization level".

When designing the optimization algorithm Li compared their algorithm to a system known as GCC that "…has more than one hundred optimization algorithms transforming programs into better forms…" (Li, 2014). GCC contains many other popular optimization algorithms within itself such as previously mentioned "loop unrolling" and "copy propagation". Yet with GCC being composed of such a large number of different algorithms, the relationships between them get complicated and can result in situations such as where an “algorithm of improving object file running speed may increases object file size”. Additionally, GCC is highly customizable, which is useful but means that “ a plan for one project can't be reused for another one because different projects have different characteristics” and developers have to constantly manually figure this out and decide the best option for their program. Some approaches to tackling this issue have used heuristic algorithms, like genetic algorithms, “to search for near optimal solutions” of GCC setups for specific projects; but currently, heuristic methods are very time consuming. Contrastingly, “machine learning is an approach with many methods including statistical methods [and it] predicts near optimal solutions in a near real-time manner”. One important aspect of the machine-learning based approach is feature design.

For machine learning approaches, “feature design is critical” (Li, 2014); and it is important to note that “program features can be divided into static features and dynamic features”. In many other machine-learning based approaches “program features are usually obtained by defining them directly by experience before learning”. This shifts the goal of the problem from “finding the best optimization algorithm combination to the problem of finding the best features of program”. The goal of the approach discussed in this paper was to try and automate some of the program feature generation through the use of templates. With the use of templates and automation of feature generation, “hundreds of features can be generated with predefined templates and the most effective ones will be selected from them with feature selection methods”. This is done by implementing “a feature extractor based on GCC which can extract program features and count their value at different phases during compilation”. This essentially allows for the program to analyze the program and generate basic features of the program from which it will search for optimal compiler optimization algorithms to use on them and provide an optimal GCC setup that quickly maximizes efficiency for a program in compilation.

For the sake of brevity, I will not be diving into the technical details of everything proposed here for each stage, I will just focus on the high-level functionality of the method provided in this research paper. Thus, some of the major features of what was done can be summed up in three basic parts:

One: Li and his team presents a "method of generating more static program features from predefined templates and selecting the most useful ones for training period and prediction period". This element plays into the machine-learning aspect of their optimization method very tightly.

Two: Li and his team note that through "different optimization phases, feature value change". By using that additional information across different phases, Li and his team designed a system that more accurately predict correlations between certain feature values and program characteristics. The machine-learning algorithm utilizes that changing data across different phases to aid in the learning model's predictions.

Three: Finally, they implemented “the prototype as mentioned above [in the third paragraph that was about feature design] based on GCC and GCC plugin system” which ended up showing that their approach was very effective.

In summary, Li and his team essentially utilized GCC – and its collection of many different customizable optimization algorithms – in tandem with a machine learning system. The machine learning system was supplemented by a statistical approach to automatically identifying and generating program features and adjusting those feature values across different optimization phases in such a way that the machine learning algorithm can identify the most optimal GCC compilation optimization setup for a specific program.

Sources

Li, F., Tang, F., & Shen, Y. (2014). Feature mining for machine learning based compilation optimization. Paper presented at the 207-214. https://doi.org/10.1109/IMIS.2014.26