International Journal of Research in Engineering, Science and Management
Volume-2, Issue-4, April-2019
www.ijresm.com | ISSN (Online): 2581-5792

A Survey of Machine Learning and Deep Learning Techniques for Compiler Optimization

K. Manasvi Bhat[1], Pratiksha P. Anchalia[2], Rushali Mohbe[3], A. Parkavi[4]
[1.,2,3]*Student, Dept. of Computer Science and Engineering, M. S. Ramaiah Inst. of Technology, Bangalore, India*
[4]*Assistant Professor, Dept. of Computer Science and Engg., M. S. Ramaiah Inst. of Tech., Bangalore, India*

***Abstract*: Designing an optimized compiler is not only difficult but also very time-consuming, especially when it has to be done manually. Recent work has proven how the design and construction can be greatly simplified and improved by leveraging machine learning approaches. Most approaches employing machine learning techniques require features of the program to be specified. Selecting the best features is crucial to increase the quality of the heuristics in machine learning and deep learning. However, even having chosen the best possible heuristic may not be enough as training the machine learning model is an expensive and a repetitive task. This paper discusses the different techniques that can be utilized to optimize and further improve the quality of the heuristics chosen and the overall quality of the machine learning and deep learning models, thus improving the efficiency of the compiler. These techniques include addressing various issues including but not limited to the phase-ordering problem, multiple evaluations of a program in an iterative approach and time taken to find the optimal heuristic.**

***Keywords*: compiler optimization, machine learning, deep learning, phase-ordering, loop unrolling, survey.**

1. Introduction

Using machine learning in order to optimize a compiler is a technique that has received considerable interest in the past decade. A compiler-writer had to manually create and fine-tune a feature for optimization and this was not an easy task as it is a complex problem and the interaction of the compiler with the rest of the architecture had to be taken into consideration as well. This time-consuming and laborious process had to be repeated for multiple heuristics and the architecture had to be changed accordingly. Hence, exploiting machine learning and deep learning techniques for the purpose of automating the optimization of the compiler is an attractive option.

In order to overcome this problem, iterative compilation was introduced. While it was an effective method of automating the heuristic generation process, the time taken was still substantial. Hence, multiple improvements have been suggested in order to further optimize the compiler by focusing on the different areas that can be improved.

Much of the research has been oriented around solving the problem of generation of a sequence of optimization passes that is optimal, termed as the phase-ordering problem. The substantial time taken by iterative compilation can be reduced by multiple techniques such as focusing the search space in iterative compilation or using active learning. Alternatively, the iterative process can be eliminated altogether by employing a deep learning model that takes the raw source code as input, or by employing an artificial neural network.

2. Literature survey

Zheng Wang et al. in their research work have described the use of machine learning techniques for compiler optimization. Their proposed system consisted of an ensemble model that included algorithms such as Support Vector Machines, Decision Tree and K-means clustering. Thus, the ensemble model constructed was a mixture of supervised and unsupervised algorithms. Their model was tested and it was found to have efficient and accurate results. Parallel program optimization is the future scope of their area of work [1].

Antoine, Monsifrot et al. in their work describe the dependency of compiler optimization on microprocessor architecture. Their work discusses a method that targets a particular microprocessor to achieve compiler optimization automatically using machine learning. The method is evaluated against the loop unrolling method. Loop unrolling also called as loop unwinding, is a technique used for loop transformation. It makes an attempt to optimize the execution speed of a program at the expense of its binary size. Decision tree algorithm and Boosting has been utilized in this piece of work. To conclude, machine learning algorithms other than decision tree can be explored to achieve loop unrolling [2].

Cummins et al. propose a model based on deep neural networks to train on raw code. The model aims at heustrics optimization and is flexible as it can solve multiple optimization problems. The predictive model is trained using supervised learning. The model aims to achieve maximum performance by learning the relation between important features and optimization decision. The proposed deep tune model extracts important features automatically and this system is evaluated against two standard techniques and it is found to have performed better than both [3].

Sameer Kulkarni et al. in their research work attempt to solve the phase ordering problem that is encountered while optimizing a compiler. The proposed method introduces an automatic approach to select a good order in which these optimizations must be performed giving rise to a dynamic

compiler. It compares the problem to a Markov process and utilizes characteristics of the current state to solve the phase ordering problem. They also propose a neuro evolution based technique to design a neural network that will perform optimization. The Neuro Evolution of Augmenting Topologies (NEAT) utilizes a neural network for phase ordering. The phase ordering problem has been explored for a long period of time and the proposed neuro evolution based approach makes a commendable contribution in addressing this problem [4].

Cong et al. in their work make an effort to study the impact of compiler optimizations on High Level Synthesis. HLS is described as a process that accepts input as source code in a high level language to generate register-transfer-level codes (RTL). The authors study the impact of phase ordering, source and intermediate register (IR) level optimizations on high level synthesis. The source level optimizations studied include loop unwinding, loop pipelining and array partitioning. The Intermediate Register (IR) level optimizations include a set of several unique optimizations. Random Search and Genetic Algorithms were implemented to study the effect of Intermediate Register (IR) level optimizations on high level synthesis. The future scope of this piece of research can include the development of a predictive model that would help foresee the effect of compiler optimizations on high level synthesis [5].

Wawrzynek et al. in their research work propose the usage of reinforcement learning agents for the purpose of addressing the phase-ordering problem. They make use of Policy Gradient and Deep Q-Network for the purpose. The various states in reinforcement learning are represented using several static features which are extracted from the intermediate representation of the LLVM program. Considering the number of clock cycles as the performance metric, it is found that this technique is 16% better in performance than the other algorithms which are currently in use [6].

Jay Patel et al. in their research work propose a technique to address the problem of applying optimization using artificial neural networks. 4Cast-XL is made use of for constructing ANNs which is integrated into Jikes RVM's optimization driver. A series of steps are applied on every method that is dynamically compiled in order to obtain the optimization technique that is best suited for the method [7].

Silvano et al. in their research work propose a technique to provide application-specific optimizations. The probability distribution of the various compiler optimization techniques is obtained by applying statistical methodology and sampling of the same is performed. The relevant statistical relations are learnt from training various applications. Bayesian networks are made use of as the statistical model. The proposed technique is found to have three times better performance than the random iterative compilation [8].

Vega-Lopez et al. propose a technique to automate the process of taking machine learning models from their prototype into deployment. The translation of machine learning models into their optimized source code is performed using a special-purpose compiler. The technique ensures a modular structure along with an efficient code which integrates seamlessly in production environments [9].

Wei, Schwartz et al. in their research work propose a new infrastructure to overcome the problems faced as a result of the existing frameworks which make use of deep learning techniques. Various techniques for the optimization of compilers are proposed to optimize the neural network computations [10].

Machine learning models have been employed in compilers to select a reasonable list of features and remove any redundant ones. Hugh Leather et al. proposes a method to generate new features by searching through the entire feature space. The feature space has been represented as a grammar wherein every sentence represents a feature. The entire feature space is searched for heuristics that can best contribute to the increase in the performance of the machine learning model [11].

In an alternate approach by F. Agakov et al., they proposed to accelerate the process of iterative compilation by automatically focussing the search on areas that would give the best performance. This method is not restricted by any search algorithm, compiler architecture or search space. Program features are used in order to focus the search when a new program is encountered. This technique has been proven to improve the search time by a significant amount, especially on large search spaces [12].

In order to reduce the time required in finding optimal heuristics, William F. Ogilvie et al. proposed employing active learning in compiler design which is capable of minimizing the cost of iterative compilation. This approach not only selects the training samples required but it also selects the number of samples required per example using sequential analysis. Such a system greatly simplifies the process of iterative compilation and makes it faster [13].

Apart from choosing the right heuristics and ensuring that the time taken to do so is not substantial, it is also necessary to build a compiler that is capable of adjusting to the modifications in the architecture. Christophe Dubach et al. addresses this challenge by employing machine learning to automatically adapt to the changes in the underlying micro-architecture. Their approach is the first step in the building of a universal compiler that can optimize a program for any underlying platform [14].

While it is important to increase the performance of the compiler, there should also be an efficient metric in order to determine the speedup of the optimized program. Christophe Dubach et al. proposed a machine learning technique that is capable of predicting the speedup of an optimized program using the features of the modified program. This model can then be used to anticipate the speedup of a program without actually executing it [15].

3. Discussions

In our survey, we identified the phase ordering problem as one of the most common problems faced while performing

compiler optimization. The problem of determining the order in which the optimization passes need to be applied in order to result in an improved performance of the generated code is termed as the phase-ordering problem. Several research papers addressed this problem with the aim of mitigating the impact of the phase-ordering problem on the efficiency and performance of the compiler. The use of artificial neural networks for addressing the problem demonstrates that profiles of programs can be used for code optimization. In comparison to it, reinforcement learning algorithms result in 16% better performance.

Loop unrolling also termed as loop unwinding is a transformation performed to increase the speed of the compiler. A loop is re-written as a sequence of similar independent statements to minimize the overhead. A compromise on the binary size has to be made in order to achieve this transformation. This problem is often clubbed with loop pipelining. In our study we observed that many researchers proposed systems and evaluated their systems against the existing loop unrolling method. They made use of several machine learning algorithms such as boosting and the decision tree algorithm. It was found that machine learning techniques gave better results than the existing techniques. Thus a variety of advanced machine learning techniques to achieve better performance must be explored.

The time that a machine learning algorithm takes to determine the best heuristics to optimize the compiler, although fast, is not fast enough. The two common techniques for addressing this problem is focusing the iterative search and employing active learning. While focusing the search would seem like the better option, it is interesting to note that with active learning, not only is the time taken to compute the heuristics reduced, but the number of training samples required is also greatly reduced, thus making is much easier to use.

In order to overcome the problems of the random iterative compilation techniques, auto-tuning techniques, which make use of Bayesian networks, provide application-specific optimization which ensuring three times faster results than random iterative compilation. The future scope of this technique can be its use in addressing the phase-ordering problem.

4. Conclusion

A survey of the research work based on compiler optimization techniques has proven that the problems in this field which were once considered to be cumbersome can now be addressed efficiently with the use of machine learning techniques. The findings of our survey helped us conclude that the use of deep learning and machine learning techniques for compiler optimization have considerably improved the performance of the compiler.

Several traditional machine-learning techniques as well as emerging techniques, such as artificial neural networks (ANN), supervised-learning based techniques, unsupervised learning based techniques and other advanced techniques have been implemented to achieve compiler optimization. Identifying the problems faced by a compiler plays a key role in choosing a method to address the same. In most cases researchers made use of an ensemble of several techniques to obtain best possible results.

This survey paper is an attempt at aggregating the findings in the field of compiler optimization using machine learning techniques. Further exploration of the advanced and emerging techniques described and a combination of these to achieve compiler optimization beyond the state-of-the-art performance of the same can be the future scope of this area of work. We aim to assist researchers compare their proposed methodologies with the existing ones that have been studied in our survey.

References

[1] Wang, Zheng, and Michael O'Boyle. "Machine learning in compiler optimization." Proceedings of the IEEE, 99 (2018): 1-23.
[2] Monsifrot, Antoine, François Bodin, and Rene Quiniou. "A machine learning approach to automatic production of compiler heuristics." in International conference on artificial intelligence: methodology, systems, and applications, pp. 41-50. Springer, Berlin, Heidelberg, 2002.
[3] Cummins, Chris, Pavlos Petoumenos, Zheng Wang, and Hugh Leather. "End-to-end deep learning of optimization heuristics." in 2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT), pp. 219-232. IEEE, 2017.
[4] Kulkarni, Sameer, and John Cavazos. "Mitigating the compiler optimization phase-ordering problem using machine learning." in ACM SIGPLAN Notices, vol. 47, no. 10, pp. 147-162. ACM, 2012.
[5] Cong, Jason, Bin Liu, Raghu Prabhakar, and Peng Zhang. "A study on the impact of compiler optimizations on high-level synthesis." in International Workshop on Languages and Compilers for Parallel Computing, pp. 143-157. Springer, Berlin, Heidelberg, 2012.
[6] Haj-Ali, Ameer, Qijing Huang, William Moses, John Xiang, Ion Stoica, Krste Asanovic, and John Wawrzynek. "Auto Phase: Compiler Phase-Ordering for High Level Synthesis with Deep Reinforcement Learning." arXiv preprint arXiv:1901.04615(2019).
[7] Patel, Jay, and Mahesh Panchal. "Code Optimization in Compilers using ANN." (2014): 557-561.
[8] Ashouri, Amir Hossein, Gianluca Palermo, and Cristina Silvano. "Auto-tuning Techniques for Compiler Optimization." (2016).
[9] Castro-Lopez, Oscar, and Ines F. Vega-Lopez. "Multi-target compiler for the deployment of machine learning models." In Proceedings of the 2019 IEEE/ACM International Symposium on Code Generation and Optimization, pp. 280-281. IEEE Press, 2019.
[10] Wei, Richard, Lane Schwartz, and Vikram Adve. "A modern compiler infrastructure for deep learning systems with adjoint code generation in a domain-specific IR." (2017).
[11] Leather, Hugh, Edwin Bonilla, and Michael O'Boyle. "Automatic feature generation for machine learning based optimization compilation." in Proceedings of the 7th annual IEEE/ACM International Symposium on Code Generation and Optimization, pp. 81-91. IEEE Computer Society, 2009.
[12] Agakov, Felix, Edwin Bonilla, John Cavazos, Björn Franke, Grigori Fursin, Michael FP O'Boyle, John Thomson, Marc Toussaint, and Christopher KI Williams. "Using machine learning to focus iterative optimization." In Proceedings of the international symposium on code generation and optimization, pp. 295-305. IEEE Computer Society, 2006.
[13] Ogilvie, William F., Pavlos Petoumenos, Zheng Wang, and Hugh Leather. "Minimizing the cost of iterative compilation with active learning." in Proceedings of the 2017 International Symposium on Code Generation and Optimization, pp. 245-256. IEEE Press, 2017.
[14] Dubach, Christophe, Timothy M. Jones, Edwin V. Bonilla, Grigori Fursin, and Michael FP O'Boyle. "Portable compiler optimization across embedded programs and microarchitectures using machine learning." in

Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture, pp. 78-88. ACM, 2009.

[15] Dubach, Christophe, John Cavazos, Björn Franke, Grigori Fursin, Michael FP O'Boyle, and Olivier Temam. "Fast compiler optimization evaluation using code-feature based performance prediction." in Proceedings of the 4th international conference on Computing frontiers, pp. 131-142. ACM, 2007.