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

A Study on Iterative Code Optimization using Machine Learning

G. R. Amith[1], K. N. Tejasvini[2], A. Parkavi[3]
[1,2]*Student, Department of computer Science, Ramaiah Institute of Technology, Bangalore, India*
[3]*Assistant Professor, Department of computer Science, Ramaiah Institute of Technology, Bangalore, India*

***Abstract*: Code optimization leads to less time and resource consumption while executing the program. Machine learning techniques are used to optimize the code. Optimization most of the time carried iteratively. Choosing the order of optimization and best approach for optimization is the issue to be addressed. Reducing the complexity or using the less resources of system such as Processor time, memory consumption and power consumption is the agenda of the optimization. We review different kinds of methods of optimization using literature survey.**

***Keywords*: optimization, machine, learning, memory, resource, processor**

1. Introduction

Creating the best iterative code optimization using machine learning techniques is not an easy task. Basically compilers have two main jobs to do 1. Translation and 2. Compilation. Reducing the complexity of programs may be by removing the duplicate declaration of variable, eliminating few lines of code which are duplicate or available by default or reducing the methods which can be compromised is basically what code optimization. There are two ways of optimization 1. Single way and 2. Multi-way optimization. These approaches are applied iteratively to get the better quality result. Reducing the complexity or using the less resources of system such as Processor time, memory consumption and power consumption is the agenda of the optimization. Naturally, better optimization leads to better system with less execution time.

For reducing code optimization, selection of good optimization technique, better order for optimizing and reducing the complexity is important. There can be two way of order of optimization - fixed order and varying order. Different Machine learning techniques are used for optimization such as Artificial Neural Network (ANN) and Genetic algorithm ext. Genetic algorithms applied on the historically available data features - to be able predict the best optimization technique.

Iterative code optimization is one of the best optimization technique. Advantage of this method is, increased optimization level while not compromising with the program. Disadvantage is Iterative optimization increases the time taken for optimization as it runs the model multiple time.

Genetic methods are used to study the past data to be able to predict the level of optimization for the present data. Until the researchers found out the automatic iterative optimization it was tedious job to optimize the compiler. In this paper, we study different Research paper on compiler optimization using machine learning approaches to gain the knowledge about the techniques and methods used for optimization by different research. By comparing different machine learning techniques how good the genetic algorithms is, can be answered.

2. Related works

A. Literature review

In our first research paper [1], Prediction technique are used to study the prior data and based on this best data point is found out for the best result possible. Machine learning can be automatic this is the advantage of machine learning in code optimization [1]. The input for this model is source program fed to a model i.e. Portable optimizing compiler which generates the best optimization passes. This method iteratively executed till the best model. Finally optimized binary code is the final output for the program. Result Evaluation method that are used to evolve this model are 1. Cross validation and 2. Best performance Achievable [1]. Advantage of this is approach it results in a best optimization technique. Disadvantage is it takes lot of time.

In “Machine learning in compiler optimization” by Wang et al, based on the prior data new data point will be predicted. There are two main important stages in a model, 1. Training data used to learn the model. 2. Model applied on recent programs [2]. Model works like this, first the source program fed into the feature optimizer. This approach is called as Feature engineering. In the second approach that is Learning model, training programs are fed into feature optimizer that will be given input as supervised machine learner. Finally, a model will be created. The learning algorithms task is to find out the correlation between feature and optimal decision. The disadvantage of this model is it takes more time compare to other machine approach as support vector machine.

In “Automatic Feature Generation for machine learning Based optimization compilation” by Leather et. al. [3], compiler optimization is automated. Quality of the features are ensured as this are important for the accuracy.

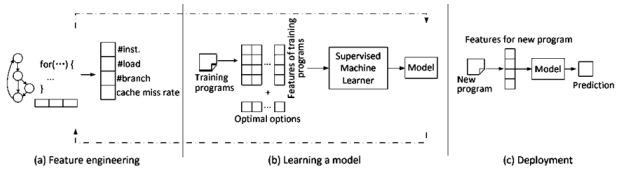

Fig. 1. A generic view of supervised machine learning in compilers

Learned heuristics adopts to new environment. Machine learning tools deals with the optimization. Feature search is an important task carried in this paper. The working methods of this approach are firstly data are generated this are passed to genetic programming search then feature values are passed to ML tool. This tool builds the list of features. However, it's tedious and time consuming task.

In "Mitigating the compiler optimization phase ordering problem using machine learning" by Kulkarni et al[4], Selecting the correct order where optimization techniques are applied is significant problem. Default optimization decreases the performance. Artificial Neural Network (ANN) are for prediction. This were induced using NEAT [4]. Issues with GA approach is, search technique are expensive as they have to evaluate different optimization orders. Solution for this is instead GAs and other expensive techniques machine learning approach can be used. Main advantage of this technique is its inexpensive.

In "Studying the influence of standard compiler optimization on symbolic execution" written by Dong et. al. [5], Symbolic execution which is time consuming technique for Path-Based analysis is used. Approaches used in the program are for conventional programming.

Compiler optimization are performed to evaluate for the performance of symbolic execution [5]. Using DFS Symbolic execution are implemented. KLEE is a symbolic execution engine built on LLVM which is the framework for compilation [5]. Finding and analyzing the determination is the goal of this paper. for you.

B. Comparative Analysis

- Artificial Neural Network (ANN) developed similar to the human way of learning the past data and predicting the result. ANN also learns from the past data and predicts the future [4]. Neuro evaluation are used to induce the ANN for NEAT [4]. Trained model of the ANN uses features to represent optimized state [4]. ANN mostly used for prediction. ANN can be used to build power models.
- In a Genetic Programming approach, firstly cost function generated. Result of Cost function evaluated [2]. Cost function are used to build an energy.
- Model for optimization. Cost function are also evaluated quality of optimization [2].
- If would like to continue the approach in next phase well performing function are kept. Then create new function using the remaining ones. This can be carried out iteratively by passing the generated function back to the evaluation phase [2]. Both ANN and Genetic algorithms are performed iteratively.
- In any micro architecture portable optimization learns best optimization approach that can be applied. Achieving the best optimization and delivering the high quality is the goal of portable compiler optimization across embedded programs. for portable optimization, genetic algorithms were used with hill climbing optimization algorithm [1].

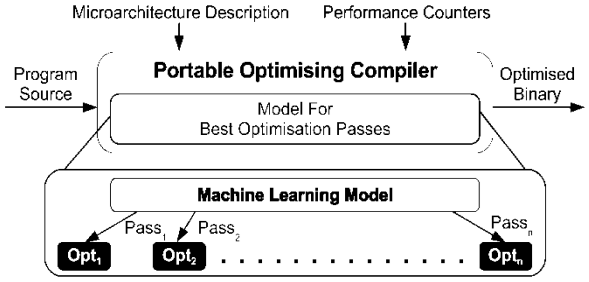

Fig. 2. Portable optimizing compiler over view [1]

- Symbolic execution has a significant influence on compiler optimization Dong et al. symbolic execution technique used for verification and testing the reliability of the software [5].

Compare to tradition optimizing techniques modern compilers for example GCC and LLVM support aggressive optimization. Most of the approaches that we see above are although different but iterative optimization used by all [5].

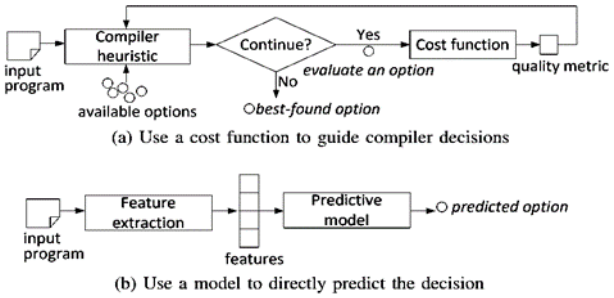

Fig. 3. Two approach for optimal decision

3. Discussion

- Most of the approaches takes in their own time. Compare to other approaches Genetic algorithms just works fine.
- Symbolic execution although test the reliability it takes its own time. Symbolic techniques contain large number of paths.
- Artificial Neural Network which predicts best optimization method by studying the past data but its slow while no hope that it can rightly predicts the best optimizing technique is not trust worthy.
- The accuracy level increase of Artificial Neural networks depends on the size of the past data available.
- Genetic algorithms other optimization such as hill climbing and optimization orchestration are resulting good performance.
- In Machine learning approaches generally face problems of

not understanding the complete working style like black box.

- ANN is not a trust worthy approach as compared. In iterative code optimization reducing usage of resources and time is the goal most of the methods that we see mostly seems to be working just fine.

4. Conclusion

This paper studies multiple approaches of machine learning for iterative code optimization. Better code Optimization leads to less resource, reduced time while executing a source program. We conducted literature survey where we studied research papers which have used different approaches for the optimization. In a comparative analysis how genetic algorithms. are better compared to Artificial Neural Network as found out.

References

[1] C. Dubach, T. M. Jones, E. V. Bonilla, G. Fursin and M. F. P. O'Boyle, "Portable compiler optimisation across embedded programs and microarchitectures using machine learning," *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, New York, NY, 2009, pp. 78-88.

[2] Z. Wang and M. O'Boyle, "Machine Learning in Compiler Optimization," in *Proceedings of the IEEE*, vol. 106, no. 11, pp. 1879-1901, Nov. 2018.

[3] H. Leather, E. Bonilla and M. O'Boyle, "Automatic Feature Generation for Machine Learning Based Optimizing Compilation," *2009 International Symposium on Code Generation and Optimization*, Seattle, WA, 2009, pp. 81-91.

[4] Kulkarni, Sameer & Cavazos, John. (2012). Mitigating the Compiler Optimization Phase-Ordering Problem using Machine Learning. ACM SIGPLAN Notices.

[5] S. Dong, O. Olivo, L. Zhang and S. Khurshid, "Studying the influence of standard compiler optimizations on symbolic execution," *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*, Gaithersbury, MD, 2015, pp. 205-215.

[6] Stephenson, Mark & Amarasinghe, Saman & C. Martin, Martin & O'Reilly, Una-May. Improving compiler heuristics with machine learning. Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI). 38. 77-90, 2003.

[7] J. Chipps, M. Koschmann, S. Orgel, A. Perlis, and J. Smith, "A mathematical language compiler," in Proceedings of 11th ACM national meeting. ACM, 1956,

[8] P. B. Sheridan, "The arithmetic translator-compiler of the ibm fortran automatic coding system," Communications of the ACM, 1959.

[9] M. D. McIlroy, "Macro instruction extensions of compiler languages," Communications of the ACM, vol. 3, no. 4, pp. 214–220, 1960.

[10] A. Gauci, K. Z. Adami, and J. Abela, "Machine learning for galaxy morphology classification,", 2010.

[11] K. D. Cooper, A. Grosul, T. J. Harvey, S. Reeves, D. Subramanian, et al. adaptive compilation made efficient. 2005.

[12] L. Almagor, K. D. Cooper, A. Grosul, T. J. Harvey, S. W. Reeves, D. Subramanian, L. Torczon, and T Waterman, "Finding effective compilation sequences," 2004.

[13] P. J. Joseph, K. Vaswani, and M. J. Thazhuthaveetil. A predictive performance model for superscalar processors. In MICRO-39, 2006.

[14] T. S. Karkhanis and J. E. Smith. A first-order super processor model. In ISCA, 2004.