Machine Learning in Compilers

Hala Ali Khan halaalikhan@gmail.com
Mahrukh Ali Khan rehmanmahrukh7175@gmail.com

January 2022

Contents

List of Figures

1 Abstract

Optimisation of compilers is a tedious task, because compilers work with a diverse range of programs and the systems that they are running on are also complex. with the passage of time as the architecture of hardware and the software are updated the optimization of compiler may not be able to work with the working environment and their techniques may become obsolete. In order to overcome this very problem machine learning was introduced in the compilers, about a decade ago. in this paper we are going to discuss about the past, present and future of compilers using various compilation techniques and how emergence and convergence of machine learning has helped compilers to mold themselves according to the changes in hardware and software. A relationship between compilers' optimization strategies and machine learning is described in this paper, which include feature engineering as well training and deployment of models. Moreover, a brief analysis on how machine learning is embedded in the compilers and an introduction to the machine learning model is also included in this research paper.

Keywords: *Machine Learning, Compiler Optimization Techniques, Iterative Compilation, Compiler tuning*

2 Introduction

Compilers are used for the conversion of human language into binary or machine language and vice versa. The source code has to go through multiple stages to be converted into a machine code. These compilers use multiple optimization techniques in order to improve their performance. In the beginning, the compilers could not choose the best optimizing technique so the developers introduced optimizing compilers. They used different command lines and this practice gave rise to iterative compilation. Later on, a better technique, known as machine learning, was introduced. Machine learning is branch of Artificial Intelligence focused at detecting and predicting order. Machine learning has been an important part in compilers and has been an emerging field. It enables compilers to optimize the code without compiling it multiple times. In this research paper, we are going to discuss the past, present and the future of compilers and how machine learning has helped in improving the performance of the compiler and the systems they are running on.

3 Literature Review

Compilers perform two jobs i.e. translation and optimization. First, source code is translated into binary language. Second, they search for the most effective translation. Performances of different translation varies significantly. The main focus is mostly on second goal which is related to the performance of the compiler. Compilers cannot choose best option on their own so multiple techniques have been introduced by developers, one after another. Iterative compilation

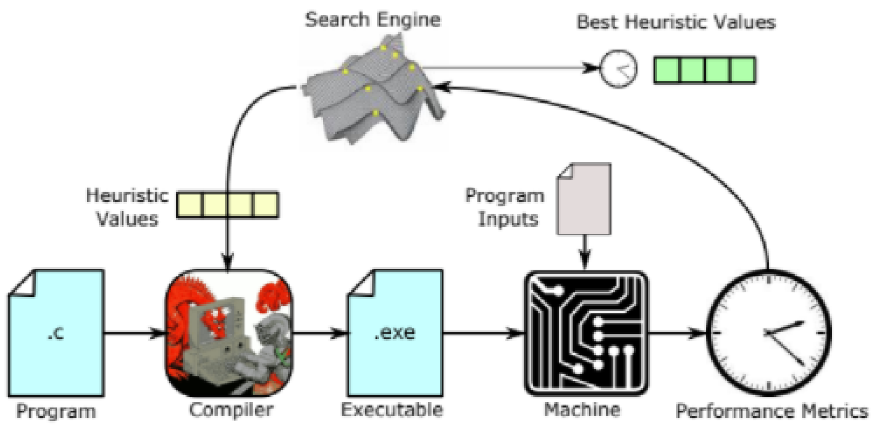


Figure 1: Iterative Compilation

was introduced in the beginning. Due to its simplicity, this technique became quite popular. It repeatedly implements a large number of optimization techniques until the best optimal combination of these techniques has been found. This technique easily accommodates changes and works on evidence rather than belief. In iterative compilation, space of optimization strategies is defined, these include techniques like loop unrolling, tilings and searches the best option among them. It compiles the program, executes it enough times and measures the performance (time required to process the code and generate output). Optimization in compilers is implemented to gain the targets such as compiler flags, phase ordering, size of code etc. Different heuristics in iterative compilation used different searching techniques such as: i. Genetic Algorithms ii. Random Search iii. Greedy approaches Simple Grid Search was used to explain the following defects: i. It was difficult for humans to extract heuristics ii. For each architecture new heuristics were needed to be introduced iii.Performance gain was quite high while selecting best optimization technique

3.1 Drawbacks of iterative compilation

The implementation of iterative compilation is costly in general computing however it is used in embedded applications where the cost can be compensated, due to the systems produced in bulk. The performance of the compiler is improved but the systems performance is affected. It takes more time which means resources are busy for a longer period of time and due to multiple compilation, more memory than usual is required. Another factor is time consumption. Due to these drawbacks, the use of iterative compilation has been limited to certain type of applications. Optimization has been studied since 1800s, the question arises what took so long for converging optimization and machine learning? The two major reasons highlighted are: (i) Due to increase in performance of hardware the software is unable to catch the same momentum, thus leading to a software gap. Compiler designers are trying to undo this gap. (ii) The evolution of the computer architecture is so quick and it brings eccentricity. The compiler designers are trying to catch up with these changes.

3.2 Machine learning in compilers

Iterative compilation has the capability to improve the performance of the program to a significant extend. Later on the developers tried to introduce a technique which optimizes the program same as iterative compilation but in a single compilation. Machine learning helped the developers to achieve this goal. Machine learning used the output data of iterative compilation, develops a training model and trains the compiler over that data. Once the compiler has been trained, a random query is given to the compiler and it predicts the possible outcome. Then the outcome is compared with the trained data. The ratio of accuracy shows the extent of compiler training. If any changes occur to the architecture, the environment of the compiler, operating system or the application domain, the training data is simply revitalized.

3.2.1 Feature Engineering

First step to learn anything useful about the programs is to categorize them. Machine learning uses quantifiable features to characterize the programs. These features may include static data structures from the source code or obtained at runtime, or even both. Algorithms used in machine learning work with fixed length of inputs. So any features are then converted into a fixed length vector. This process, in which the features are tuned or summarized to a fixed length is called feature engineering. This process needs to be performed iteratively in order to gain set of best tuned features, which ultimately helps in formulating a machine learning model.

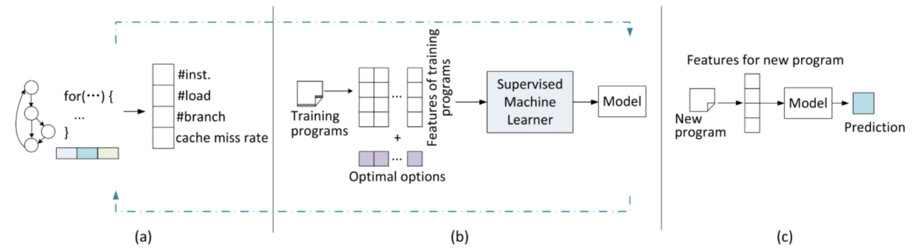


Figure 2: Machine Learning in compilers

3.2.2 Model learning

After feature engineering, next step is to formulate a learning model and enfolding it to a machine learning algorithm. In this scenario, instead of gathering data from any other outside source, data from the already existing application is collected and some representative programs of that application domain are selected. Each of the selected program has to go through a set of processes to be qualifying as model data. First of feature engineering is done and the values are calculated after that best optimization technique is selected via applying different techniques and compiling the program multiple times. The calculated feature values and the optimal technique, for each program, is then added to the machine learning algorithm. The algorithm then enfolds this information automatically generates a model. The main task of learning model is to find the relation between features and their corresponding optimization techniques. Then comes testing of the model, where the model predicts the best optimization technique against given set of features. Process of feature engineering and data training has to be done multiple as the performance of the model greatly depends upon these two features.

3.2.3 Deployment

In this final step the learned model is deployed to the compiler to predict best optimization technique for the new program. The compiler first extracts the features from the new input program then sends these features to the learned model for prediction. Model building is not a tedious task and this feature gives an upper hand to machine learning over other optimization techniques, iterative compilation being one of them. This feature makes machine learning a flexible strategy as it can be accommodated easily even if the hardware architecture, application domain or operating system of the machine is changed.

4 Analysis

To transform the written code into right sequence is one of the main challenges for compilers. The sequence of those alterations in also an important section to be focused. In order to achieve this goal evaluation of different compilation options takes place. One of the approach is to implement each compilation option and after that design a program which gathers the related performance dimensions. This approach is known as auto-tuning or iterative compilation, in other words. Due to loopholes in this compilation optimization techniques two other techniques are used. In the first technique a cost function is designed and is used as proxy to calculate the accuracy of compiler's decision. The other option offers direct prediction of best option.

4.1 Introducing a cost function

To calculate the quality of possible compilation techniques a compiler makes use of cost function. The cost function itself works on some quality matric, which can be size of the code, time required to execute code or energy and resource consumption etc. the basic function of cost function is to evaluate the possible compilation option and then select the best one. It saves the compiler from running the program with each available technique.

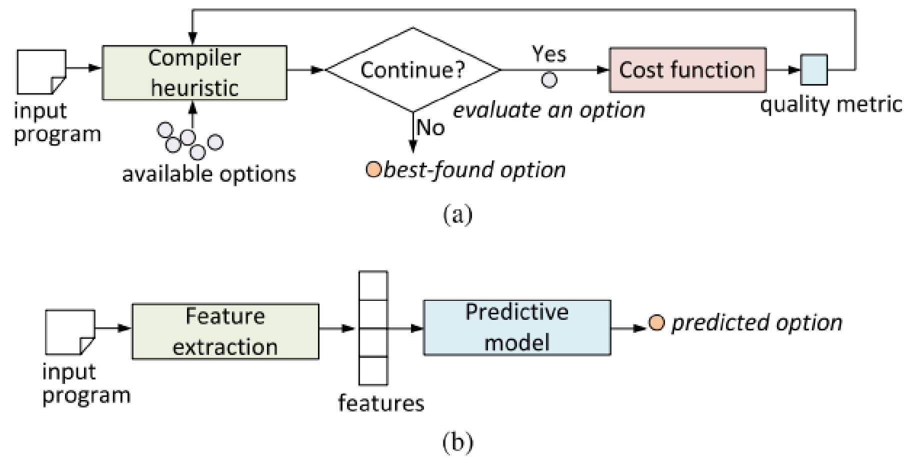


Figure 3: Optimal Compiler Decision

4.1.1 Problems in manually crafted cost function

Conventionally, a cost function is manually designed. For explanation, let us take an example of inline function. Implementation of inline function involves different metrics like number of instruction to be copied from the function, size of stack and caller after calling the function and the obtained results are feasible and comply with the adjusted threshold. The main drawback is that the importance or the weights of the dimensions are adjusted by the developer on the bases of his experience or prior knowledge. For this purpose two strategies are used:
• Trial-and-error
• One-size-fits-all

4.1.2 Performance cost function

Another way of tuning a cost function is to evaluate the execution time of the targeted program. This kind of technique uses algorithms to calculate the time of the execution of a program of a given size. This information helps the compiler to select the best option in terms of performance and hence tunes the cost function.

4.1.3 Energy consumption cost function

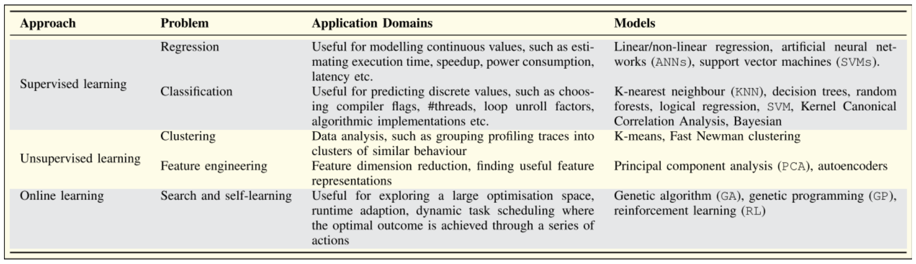
Along with performance, another important body that searches for the techniques to design an energy model to optimize software and architecture design of hardware is also working. This type of modeling usually makes use of linear regression because the readings are the continuous real values.

4.2 Directly predicting best option

Cost function may be useful for evaluating and predicting best compiling options but the overhead makes it less competitive. Therefore, researchers carried out numerous researches to develop a machine learning techniques to directly predict the best option. Initially a technique was used which was similar to decision-tree approach but it only gave result in binary decision. For example will it be okay to unroll a loop or not but it did not provide any knowledge regarding how many times the loop should be unrolled. This drawback was later taken into account and after further researches advancements were made in this area as well. Loop unroll factor helped in achieving this goal.

4.3 Machine Learning Models

Here, we are going to discuss different types of machine learning models. Two major Machine Learning techniques used in compilers are supervised and unsupervised learning.



Approach	Problem	Application Domains	Models
Supervised learning	Regression	Useful for modelling continuous values, such as estimating execution time, speedup, power consumption, latency etc.	Linear/non-linear regression, artificial neural networks (ANNs), support vector machines (SVMs).
	Classification	Useful for predicting discrete values, such as choosing compiler flags, #threads, loop unroll factors, algorithmic implementations etc.	K-nearest neighbour (KNN), decision trees, random forests, logical regression, SVM, Kernel Canonical Correlation Analysis, Bayesian
Unsupervised learning	Clustering	Data analysis, such as grouping profiling traces into clusters of similar behaviour	K-means, Fast Newman clustering
	Feature engineering	Feature dimension reduction, finding useful feature representations	Principal component analysis (PCA), autoencoders
Online learning	Search and self-learning	Useful for exploring a large optimisation space, runtime adaption, dynamic task scheduling where the optimal outcome is achieved through a series of actions	Genetic algorithm (GA), genetic programming (GP), reinforcement learning (RL)

Figure 4: Machine Learning Models

4.3.1 Supervised Learning

In supervised machine learning a labeled set of data and important features are used for training a model. The model then learns the correspondence and relationship between the features and optimization technique that results in optimal solution. Best optimization decisions are made on the basis of these relationships learned by the model.

4.3.2 Unsupervised Learning

Another type of learning is unsupervised learning where out is not labeled. It only trains its model on the features or input data. Clustering is an example of unsupervised learning where data is grouped into several sets.

4.3.3 Reinforcement Learning

Reinforcement learning is the type of online learning where the model learns from interaction or feedback. This algorithm tries to learn on its own by the reward or feedback given to it on producing a certain result or output.

5 Conclusion

Machine learning is bringing a notable change in compiler optimization and have a significant influence on it. It will continue to do so in the coming years. In the research area of compiler, machine learning based compilation is very popular now a days. Over the last decade huge work had been done on this part of area in the form of research papers. In this research paper, we have tried to outline the overall survey of main research areas and have predicted its future direction.

6 References

@inproceedingsleather2020machine, title=Machine learning in compilers: Past, present and future, author=Leather, Hugh and Cummins, Chris, booktitle=2020 Forum for Specification and Design Languages (FDL), pages=1–8, year=2020, organization=IEEE

6.1 Reference Research Papers

@articlewang2018machine, title=Machine learning in compiler optimization, author=Wang, Zheng and O'Boyle, Michael, journal=Proceedings of the IEEE, volume=106, number=11, pages=1879–1901, year=2018, publisher=IEEE

@inproceedingschen2010evaluating, title=Evaluating iterative optimization across 1000 datasets, author=Chen, Yang and Huang, Yuanjie and Eeckhout, Lieven and Fursin, Grigori and Peng, Liang and Temam, Olivier and Wu, Chengyong, booktitle=Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation, pages=448–459, year=2010

@inproceedingsbodin1998iterative, title=Iterative compilation in a non-linear optimisation space, author=Bodin, François and Kisuki, Toru and Knijnenburg, Peter and O'Boyle, Mike and Rohou, Erven, booktitle=Workshop on Profile and Feedback-Directed Compilation, year=1998

@inproceedingsbeaty1991genetic, title=Genetic algorithms and instruction scheduling, author=Beaty, Steven J, booktitle=Proceedings of the 24th annual international symposium on Microarchitecture, pages=206–211, year=1991

@articlealmagor2004finding, title=Finding effective compilation sequences, author=Almagor, Lelac and Cooper, Keith D and Grosul, Alexander and Harvey, Timothy J and Reeves, Steven W and Subramanian, Devika and Torczon, Linda and Waterman, Todd, journal=ACM SIGPLAN Notices, volume=39, number=7, pages=231–239, year=2004, publisher=ACM New York, NY, USA

@inproceedingsagakov2006using, title=Using machine learning to focus iterative optimization, author=Agakov, Felix and Bonilla, Edwin and Cavazos, John and Franke, Björn and Fursin, Grigori and O'Boyle, Michael FP and Thomson, John and Toussaint, Marc and Williams, Christopher KI, booktitle=International Symposium on Code Generation and Optimization (CGO'06), pages=11–pp, year=2006, organization=IEEE

@inproceedingspan2006fast, title=Fast and effective orchestration of compiler optimizations for automatic performance tuning, author=Pan, Zhelong and Eigenmann, Rudolf, booktitle=International Symposium on Code Generation and Optimization (CGO'06), pages=12–pp, year=2006, organization=IEEE

@articlekulkarni2004fast, title=Fast searches for effective optimization phase sequences, author=Kulkarni, Prasad and Hines, Stephen and Hiser, Jason and Whalley, David and Davidson, Jack and Jones, Douglas, journal=ACM SIGPLAN Notices, volume=39, number=6, pages=171–182, year=2004, publisher=ACM New York, NY, USA

@inproceedingssteuwer2017lift, title=Lift: a functional data-parallel IR for high-performance GPU code generation, author=Steuwer, Michel and Remmelg, Toomas and Dubach, Christophe, booktitle=2017 IEEE/ACM International Symposium on Code Generation and Optimization (CGO), pages=74–85, year=2017, organization=IEEE

@inproceedingsleather2009raced, title=Raced profiles: efficient selection of competing compiler optimizations, author=Leather, Hugh and O'Boyle, Michael and Worton, Bruce, booktitle=Proceedings of the 2009 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems, pages=50–59, year=2009

@articlefursin2014collective, title=Collective mind: Towards practical and collaborative auto-tuning, author=Fursin, Grigori and Miceli, Renato and Lokhmotov, Anton and Gerndt, Michael and Baboulin, Marc and Malony, Allen D and Chamski, Zbigniew and Novillo, Diego and Del Vento, Davide, journal=Scientific Programming, volume=22, number=4, pages=309–329, year=2014, publisher=IOS Press

@inproceedingsansel2014opentuner, title=Opentuner: An extensible framework for program autotuning, author=Ansel, Jason and Kamil, Shoaib and Veeramachaneni, Kalyan and Ragan-Kelley, Jonathan and Bosboom, Jeffrey and O'Reilly, Una-May and Amarasinghe, Saman, booktitle=Proceedings of the 23rd international conference on Parallel architectures and compilation, pages=303–316, year=2014

@inproceedingsnugteren2015cltune, title=CLTune: A generic auto-tuner for OpenCL kernels, author=Nugteren, Cedric and Codreanu, Valeriu, booktitle=2015 IEEE 9th International Symposium on Embedded Multicore/Many-core Systems-on-Chip, pages=195–202, year=2015, organization=IEEE

@inproceedingsgrewe2013portable, title=Portable mapping of data parallel programs to opencl for heterogeneous systems, author=Grewe, Dominik and Wang, Zheng and O'Boyle, Michael FP, booktitle=Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO), pages=1–10, year=2013, organization=IEEE

@inproceedingsnamolaru2010practical, title=Practical aggregation of semantical program properties for machine learning based optimization, author=Namolaru, Mircea and Cohen, Albert and Fursin, Grigori and Zaks, Ayal and Freund, Ari, booktitle=Proceedings of the 2010 international conference on Compilers, architectures and synthesis for embedded systems, pages=197–206, year=2010

@inproceedingscummins2017end, title=End-to-end deep learning of optimization heuristics, author=Cummins, Chris and Petoumenos, Pavlos and Wang, Zheng and Leather, Hugh, booktitle=2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT), pages=219–232, year=2017, organization=IEEE

@inproceedingsbrauckmann2020compiler, title=Compiler-based graph representations for deep learning models of code, author=Brauckmann, Alexander and Goens, Andrés and Ertel, Sebastian and Castrillon, Jeronimo, booktitle=Proceedings of the 29th International Conference on Compiler Construction, pages=201–211, year=2020

@inproceedingshaj2020neurovectorizer, title=NeuroVectorizer: End-to-end vectorization with deep reinforcement learning, author=Haj-Ali, Ameer and Ahmed, Nesreen K and Willke, Ted and Shao, Yakun Sophia and Asanovic, Krste and Stoica, Ion, booktitle=Proceedings of the 18th ACM/IEEE International Symposium on Code Generation and Optimization, pages=242–255, year=2020

@inproceedingshuang2019autophase, title=Autophase: Compiler phase-ordering for hls with deep reinforcement learning, author=Huang, Qijing and Haj-Ali, Ameer and Moses, William and Xiang, John and Stoica, Ion and Asanovic, Krste and Wawrzynek, John, booktitle=2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pages=308–308, year=2019, organization=IEEE

@inproceedingsmirhoseini2017device, title=Device placement optimization with reinforcement learning, author=Mirhoseini, Azalia and Pham, Hieu and Le, Quoc V and Steiner, Benoit and Larsen, Rasmus and Zhou, Yuefeng and Kumar, Naveen and Norouzi, Mohammad and Bengio, Samy and Dean, Jeff, booktitle=International Conference on Machine Learning, pages=2430–2439, year=2017, organization=PMLR