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

Compiler Optimization using Artificial Intelligence

Kiran[1], H. K. Kiran Raj[2], Punith C. Jigali[3], R. Rahul[4], A. Parkavi[5]
[1,2,3,4]*Student, Department of CSE, MSRIT, Bangalore, India*
[5]*Assistant Professor, Department, MSRIT, Bangalore, India*

***Abstract*: In the recent years, the scale of applications has been increasing. Artificial Intelligence is emerging and has been used by many researchers in the field of compilers. It can also be used to optimize these large-scale applications which require high performance for critical tasks. Many existing compilers have various options to optimize the code. These can be specified by the use of flags but such implementations would vary for each CPU architecture. This would demand a lot of time and effort to provide compilers for a multitude of microprocessor architectures. A significant reduction in development time and effort can be made with the use of machine learning models. This paper explores the various methods to implement such compiler optimizations, varying from the use of fully-Bayesian generative networks to Markov Logic Networks. Many open source APIs have been used in the field of compiler optimization.**

***Keywords*: compiler optimization, artificial intelligence, machine learning.**

1. Introduction

Compilers have been developed since the 1950s and the theory behind their construction has not changed much. Besides being used for translation of programming languages from one form to another, compilers are also used for optimization of the source programs. In the recent years, many researchers have been looking into the use of AI and machine learning to improve the compilation process. As the compiler is a complex expert system, and expert systems are a branch of AI, the use of AI in optimizing a compiler is justified. Performance critical programs have been hand optimized for a speed up from a long time but the same has been achieved using ML algorithms which look for an optimal solution in large search spaces. This paper discusses the previous work of such ML driven compiler optimization. There has been such research related to optimization in the field of embedded systems, multi-core processors and deep learning. There are four sections in the paper. In the literature survey section, we discuss the various research works related to compiler optimization using AI.

2. Literature survey

Braun et al. proposed a formal model which is built with first order constructs (i.e. lifted variable elimination for single queries as well as First order knowledge compilation based on the weighted model counting. AI fields and machine learning require efficient inference algorithms. Modelling in real world scenario yields large probabilistic models. Lifting uses symmetries in the model to eliminate variables and increase the speed of compilation. First order junction tree is built to handle multiple queries efficiently using lifted variable elimination. Lifted junction tree is used to build a first order junction tree, cluster the model into sub models that contains all the information for a query after propagating information. Lifted junction tree is very efficient and has linear complexity. All operators have pre-post conditions to ensure a result does not change highly. The lifted junction tree, lifted variable elimination and First order knowledge compilation are fused to get efficient models. The fused model increases the efficiency as it speeds up the processes [1].

Stephenson, M. et al. proposes the meta optimization technique which uses the machine learning to automatically search the space of compiler heuristics. This technique is used to reduce the compiler design complexity by heuristic tuning. The proposed system uses genetic algorithm which is an evolutionary, adaptive algorithm to find compiler heuristics and it is very effective. With this technique disparity is noticed among the performance of training set and the cross validation. Overfitting on training set occurs in some of the test cases. [2]

Pan, Z. et al. proposes a fast and effective algorithm called Combined Elimination (CE) which is fast and effective orchestration of compiler optimizations for automatic performance tuning. Optimization of compile time improves the program performance and degradations. The solution is to develop dynamic, feedback-directed optimization orchestration algorithms to achieve the best program performance by automatically searching for the combination of optimization techniques. But the challenge is developing an orchestration algorithm with minimal time complexity. Some of the Orchestration algorithms are

- Exhaustive Search
- Batch Elimination
- Iterative Elimination
- Combined Elimination
- Optimization Space Exploration
- Statistical Selection

With an aim to find an optimal point in a high definition space $S = F1 \times F2 \times \ldots \times Fn$. Two metrics they have used for algorithm comparison are performance and tuning time. With the help of this they proposed a new compiler optimized algorithm called combined elimination. It optimizes the program automatically with a high dynamic tuning performance. [3]

Patel J. et al. proposes Code Optimization in Compilers using ANN. Here they developed a system which selects the best optimization orderings automatically on a per method basis by including profiles of programs within a dynamic compiler. Optimization techniques are mainly classified into machine dependent and machine independent. In machine independent we are not considering compilers and CPU attributes but we consider in machine dependent.

Some of the optimization techniques reviewed in this paper are

- Code motion
- Reduce in strength
- Loop unrolling
- Instruction level parallelism

The proposed method is to implement 4Cast-XL which is dynamic compiler. It constructs ANN and integrates into Jikes RVM optimization driver and then at the task of phase-ordering optimizations ANN is evaluated. Finally, below steps are performed repeatedly.

- Generating a feature vector for a current method's state
- Generating a profiles of the program
- Predict the best optimization to apply by using ANN.

Now run the benchmarks obtain feedback for 4Cast-XL and record execution time for each. Finally, speedup is obtained by normalizing the obtained time for each benchmarks. Results of the experiments show that the obtained profiles of the program can be used for code optimization. [4]

Dubach, C. et al. describe a machine learning driven approach to optimize a portable compiler. The paper proposes an adaptive compiler which tunes itself to an ever-changing microarchitecture. Each program would be optimized for a specific generation of microprocessor by the compiler. They generate the training data for the model using the sets of optimization passes and previous program/microarchitecture pairs. These pairs comprise 11 performance counters, Arithmetic Logic Unit, Multiplier-ACcumulator and Shifter usage, IPC, Decoder, Register file, Branch prediction access rates and Instruction cache and Data cache access and miss rates. The ML problem stated is finding the best model which outputs a mapping from the training data to a set of optimal optimization passes. They use the embedded benchmark suite, MiBench to evaluate their model on 35 programs. There is considerable improvement by using the proposed ML model over the gcc -O3 flag. [5]

Singh et al. proposed an idea on the possible relational database errors, as majority of scientific and commercial data is stored in the relational database. Database may have errors, and error checking, finding missing values in database becomes necessary. User needs to specify all the probabilistic dependencies in the data. Relational database schemata are viewed as programs that describes the probabilistic dependencies that exists in the data. The model construction for the domain experts is done to simplify the task. Using the schema given a customized fully-Bayesian generative graphical model is generated. A Bayesian graphical model is created and one can perform inference on it. Single table A is constructed with attributes xA. The distribution is generated using the type of attribute. Gaussian is for real-valued, discrete for categorical, and Bernoulli for Bool values. Foreign Component links are generated for element xA to fiB if there are foreign key attributes and the model built is used to detect the uncertainty of the foreign keys, missing values in the database. Various experiments are performed on the User-Movie-Rating database, on which the join operation is performed over all the tables. As the result all the dependencies will be lost. By treating the proportion of the cells as missing synthetic data is created. It was found that Bayesian model allows to identify the missing values, detect outliers, visualize clustering, etc. effectively [6].

Flores et al. proposed an idea for incremental compilation of the Bayesian networks. Secondary structure called junction tree is built to carry out inference on probability propagation in Bayesian networks. In Incremental compilation, only those parts of the joint tree are recompiled which might have been affected by the network's modification. A technique called as Maximal prime subgraph decomposition is used to determine the minimal subgraphs that is to recompiled and thereby be replaced with new subtrees. As incremental compilation is used, the time consumed is very much high. Other than this incremental compilation is very efficient than actual compilation. The main aim is to build the join tree (junction tree) avoiding triangulation. Incremental compilation identifies the parts of the joint tree that are affected by the Bayesian networks, and adds new sub structures into the original join tree. Various issues of incremental compilation were found to be adding an arc, removing an arc, adding a node, removing node. Using MSP the join tree affected mostly by the Bayesian network is identified and recompiled, and this algorithm also makes sure there is no triangulation. This method saves time when Bayesian network changes frequently [7].

Casado, M.L. et al. proposes the improvisation of inference compilation for probabilistic programming in scientific simulators. They have considered Bayesian inference problem in the probabilistic model family. They are working on inference compilation which makes use of universal probabilistic programming and deep learning methods. They have also introduced the probabilistic programming library based on C++ called CPProb. CPProb is developed to boost the large scale simulation code along with inference compilation which is probabilistic programming library written in C++. It exports mainly three functions: sample, observe and predict. The algorithm used is importance sampling in conjunction with inference compilation. This work is a first step to implement a probabilistic programming approach in a physical science which helps in simulation methods [8].

Hershey et al. proposed an idea to unify a number of models from machine learning, statistical text processing, vision, bioinformatics by using probabilistic graphical models. Usually the models have large dimensionalities. Efficiency can be increased by dividing the problem into series of logically independent components. Modularity helps to divide the code. Dimple is an open source API. This API uses modular architecture to make development of new inference. Dimple architecture is very easy for anyone to add new inference, it serves as front-end and compiler for hardware inferences. Another way of increasing the speed is to avoid the loop. Dimple avoids this looping as it has ability to create a large n-dimensional collections of variables as well large number of factors without loops. GP5 is one of the optimized to accelerate inference on graphs with large factor tables [9].

Radovilsky et. al. proposed an idea to develop the approximation algorithms for selection of optimal set of measurements under the dependency structure modeled by Bayesian Networks (tree-shaped). As there are two diagnostic systems, tests and hypothesis with the statistical dependencies with among the variables. The main aim of the paper [10] is to optimize the objective function, which is a tough problem in the general cases. Observation subset selection is the restricted version of the problem, as all the measurements must be selected in advanced. Various methods for to tackle these problems have been proposed. Quality of the result can be predicted by measuring the amount of allocated time. A statistical model called performance profile can be employed for this prediction. Algorithms are proposed to compute the generalized local compilation, oss in Bayesian networks. In this paper the concept of CPP is extended and presented an efficient technique for the compilation of the composite system [10].

As we know that compilers are used for converting code from high level language to machine readable format. Whereas machine learning is used for some sort of prediction. As compiler provides optimized solution among the many. But the job of finding optimal solution is very difficult. Machine learning can provide best outcome through prediction. Zheng Wang et al. proposed a method in which feature of Machine Learning can be used for optimizing compiler. This may helpful for the compiler to gain high performance. First we create a training model which consists of the different programming applications. Then for each training program we find the feature values and we try to compile the training programs through different optimization possibilities, then we store the best compiler optimized option as well. Then we try to use this whole training set with the machine learning algorithm which will result in creation of a model [11].

This model can be further used for finding best optimization option for the compiler which will increase the compiler's performance. In this manner combining of compiler optimization with machine learning will help in getting the best optimized option.

Hugh Leather et al. have examined the state of the compiler at the beginning. This result has the structure of our program code i.e. AST (Abstract Syntax Tree) and the steps of compilation. After the above step we use some ML tools to set the vectors such as number of branch counts in our program, loop levels, etc. All these terms are known as FEATURES. Then the compiler runs each and every computer program to generate a training set and stores the features for each Benchmark (executing program). This technique does the execution of computer programs iteratively to get the best Feature values. Then the compiler transfers all the computer programs set along with the best features value to the Machine Learning tool which will construct a model. Then the new benchmarks are tested against this model to get the best Compiler optimization results. In this way the Machine learning's best prediction is used for utilizing the best compiler optimization results. This further result in high performance compiler [12].

The goal is to use machine learning which has the ability to decrease number of executions. Grigori Fursin et al. proposed a method in which the compiler learns by itself to optimize computer applications/programs/benchmarks. In this project we are working with the GCC compiler, which is open source. This compiler has more advantageous features in terms of optimization when compared to other compiler. This method goes through 2 stages: one is Training of dataset and other is the Deployment in which the testing benchmarks/executable computer programs are compiled with GCC compiler. In the Training stage, gathering of different computer programs will be done along with the program structures like AST. Accuracy of building best model will be proportional to the number of benchmarks we gather along with the program structure information. To generate training examples, we use a tool named CCC (Continuous Collective Conformation framework) which will be helpful in storing execution time of benchmarks, size of the program, etc. Deployment, after collecting the sufficient training data a model is constructed using ML algorithm. This constructed model will be used for prediction of good optimization results of the GCC compiler [13].

Choosing the best optimization options for compiler is a critical task. Sameer Kulkarni et al. proposes a method to select the best optimization phases for separate sections of the same program/benchmark rather than applying for the whole program. Our approach will determine the best optimization phase ordering on a per-method-basis. To do this we use Artificial Neural Network (ANN) to predict the optimization order. This technique's training section takes different program's methods and their current state of optimization phase as an input and predicts the best optimization phase by changing some properties of the method. This technique resolves phase ordering problem by taking the benefit of Markov-property [14].

Ganapathi, Archana et al. presents a machine learning approach to optimize the compilation suitable for multicore processors. The authors discuss the drawbacks of using autotuning: 1- The size of the parameter space to explore (Search spaces with 40 million configurations need about 180 days to search). 2- Auto-tuners only try to minimize overall runtime. These drawbacks are overcome using statistical machine learning (SML) algorithms by drawing in inferences from automatically constructed models of large quantities of data. The authors consider a recent SML algorithm, called Kernel Canonical Correlation Analysis (KCCA) to identify the relationships between the set of optimization parameters and a set of resultant performance metrics to explore the search space. They conduct experimental runs to optimize Stencil code on Intel Clovertown and AMD Barcelona processors. The paper concludes that the KCCA was able to optimize two stencil codes on two multicore architectures, upto 18% over the level of a human expert [15].

Delgrande, James P. et al. describe a system, called PLP, which compiles ordered logic programs into standard logic programs. It is implemented using the Prolog programming language. It is front-end system for the logic programming systems dlv and smodels. PLP is an efficient translator as the resultant logic programs are polynomial in the size of the input program. The output of the logic programs is an answer set. The user can use the compiler by writing programs in ordered (or set-ordered) logic which will be translated to standard logic [16].

Agakov, Felix et al. propose a way to speed up the iterative compiler optimization using machine learning. The methodology uses the source code features to correlate the program to be optimized with previous knowledge in order to focus the search. The research is centered around embedded applications where performance is critical. Previous methods are slow as the search space is large and performance improvements need a large number of evaluations. The authors propose that the search can be sped up using a focused search. They conducted their experiments on a TI C6713 (a high-end floating-point digital signal processor) and an AMD Alchemy Au1500 processor (an embedded SoC processor using MIPS32 core (Au1)). The ML Search algorithms used to search the source-level transformation spaces are a blind random search and a "smarter" genetic algorithm. The authors make an assumption, using the Independent identically distributed (IID) model, that all transformations are mutually independent neglecting the effect of interactions among transformations. They present a one-off training/learning phase to build a model which is then applied to each new program [17].

Kazemi, Seyed Mehran et al. answer the two issues: 1- Kazemi and Poole (2016) compared end-to-end (compiling to a target circuit and reasoning with the circuit) run-times of their work with Van den Broeck et al. (2011)'s weighted first-order model counting, leaving the question of where exactly the speedup comes from, and 2- the actual reason behind the speedup gained by compiling to a program instead of a data structure remained untested. The authors conduct their experiments on Markov logic networks (MLNs). They explain the LRC2CPP algorithm for compiling an MLN into a C++ program. LRC2CPP is a recursive algorithm which takes as input an MLN M and a variable name vname, and outputs a C++ code which computes Z(M) and stores it in a variable called vname. Their experiment indicates that the speedup in LRC2CPP is mostly due to the reasoning with a low-level program instead of a data structure. They also explored why reasoning with a low-level program is more efficient than reasoning with a data structure. This was done by designing an implementation-independent experiment using which they tested and validated Kazemi and Poole (2016)'s hypothesis stating that low-level programs can be compiled and optimized, while reasoning with a data structure requires a virtual machine to interpret the computations, and compilers are known to be faster than interpreters [18].

3. Proposed method

We've decided to use the GCC compiler for our idea. The reason being, this particular compiler is open source, can be used for compiler optimization as it has many advantageous features which could be helpful in optimization. Along with the type of compiler we decided to do phase level optimization, this idea would be helpful in deeper optimization when compared to the whole benchmark's optimization result. We segregate data about different optimization ressults to build a training model. The result stored will be the best among all different optimization possibilities. Training dataset also stores the each and every benchmark's execution time, size of the program. To do all this process we use Continuous Collective Confirmation framework tool. After gathering all these, we test on new benchmarks and the prediction's outcome will be the best compiler optimization.

4. Conclusion

We have presented many methods to optimize the compiler performance. The AI model structure changes frequently depending on the data. The presented methods compile the data in efficient way and saves the time, such as incremental compilation that compiles only the changing structure and other remain unaffected, and thus saves time. There are open source API's that acts as front end as well as compiler for hardware. The program's execution time, size, various optimization results will be collected and we build a training model. This model will be helpful in finding the best optimization parameters for new AI structures and various programs. We will try to incorporate the presented methods with other methods for increasing the efficiency, so that parallelism and caching can be improved to speed up the runtime.

References

[1] Braun, Tanya, and Ralf Möller. "Fusing First-order Knowledge Compilation and the Lifted Junction Tree Algorithm." Joint

German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz). Springer, Cham, 2018.

[2] Stephenson, M., Amarasinghe, S., Martin, M., & O'Reilly, U. M, Meta optimization: improving compiler heuristics with machine learning. In ACM SIGPLAN Notices, Vol. 38, No. 5, pp. 77-90, June 2003.

[3] Pan, Z., & Eigenmann, R. (2006, March). Fast and effective orchestration of compiler optimizations for automatic performance tuning. In Proceedings of the International Symposium on Code Generation and Optimization (pp. 319-332). IEEE Computer Society.

[4] Patel, J., & Panchal, M. (2014, May). Code Optimization in Compilers using ANN. In Monthly Journal of Computer Science and Information Technology, Vol. 3, Issue. 5. (pp. 557-561). IJCSMC.

[5] Dubach, C., Jones, T. M., Bonilla, E. V., Fursin, G., & O'Boyle, M. F. (2009, December). Portable compiler optimisation across embedded programs and microarchitectures using machine learning. In Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (pp. 78-88). ACM.

[6] Singh, Sameer, and Thore Graepel. "Compiling relational database schemata into probabilistic graphical models." arXiv preprint arXiv:1212.0967 (2012).

[7] Flores, M. Julia, José A. Gámez, and Kristian G. Olesen. "Incremental compilation of Bayesian networks." Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence. Morgan Kaufmann Publishers Inc., 2002.

[8] Braun, Tanya, and Ralf Möller. "Fusing First-order Knowledge Compilation and the Lifted Junction Tree Algorithm." Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz). Springer, Cham, 2018.

[9] Hershey, Shawn, et al. "Accelerating inference: towards a full language, compiler and hardware stack.", 2012.

[10] Radovilsky, Yan, and Solomon Eyal Shimony. "Observation subset selection as local compilation of performance profiles.", 2012.

[11] Zheng Wang and Michael O’Boyle. Machine Learning in Compiler Optimisation.

[12] Hugh Leather, Edwin Bonilla, Michael O’Boyle. Automatic Feature Generation for Machine Learning Based Optimizing Compilation.

[13] Grigori Fursin, Cupertino Miranda, Olivier Temam, Mircea Namolaru, Elad Yom-Tov, Ayal Zaks, Bilha Mendelson, Edwin Bonilla, John Thomson, Hugh Leather, et al. MILEPOST GCC: machine learning based research compiler.

[14] Sameer Kulkarni, John Cavazos. Mitigating the Compiler Optimization Phase-Ordering Problem using Machine Learning.

[15] Ganapathi, A., Datta, K., Fox, A., & Patterson, D. (2009, March). A case for machine learning to optimize multicore performance. In Proceedings of the First USENIX conference on Hot topics in parallelism (pp. 1-1). Berkeley, CA: USENIX Association.

[16] Delgrande, J. P., Schaub, T., & Tompits, H. (2000). A compiler for ordered logic programs. arXiv preprint cs/0003024.

[17] Agakov, F., Bonilla, E., Cavazos, J., Franke, B., Fursin, G., O'Boyle, M. F., & Williams, C. K. (2006, March). Using machine learning to focus iterative optimization. In Proceedings of the international symposium on code generation and optimization (pp. 295-305). IEEE Computer Society.

[18] Kazemi, S. M., & Poole, D, Why is Compiling Lifted Inference into a Low-Level Language so Effective?, 2016.