International Journal of Research in Engineering, Science and Management
Volume-2, Issue-5, May-2019
www.ijresm.com | ISSN (Online): 2581-5792

A Survey of Machine Intelligence Heuristics in Modern Compiler Design

Miskin Dash[1], Lakshya Sharma[2], Mridul Tiwary[3], A. Parkavi[4]
[1,2,3]*Student, Dept. of Computer Science & Engineering, M. S. Ramaiah Institute of Technology, Bangalore, India*
[4]*Assistant Professor, Dept. of Computer Science & Engg., M. S. Ramaiah Inst. of Technology, Bangalore, India*

***Abstract*: This survey paper aims to illustrate the evolution in techniques for optimization of compilers. Optimization of compilers is a crucial task, and is often the most time expensive criteria. With the plethora of different compilers available for the same applications, it is paramount to use the best trade off available to us. This trade off can be in terms of cost or compilation time. Compiler suites that are comprehensively optimized offer a nimiety of benefits to developers and consumers alike, producing code almost 20-30% faster than standard benchmarks. It is beneficial to the user as well, extending lower bandwidth and catering to multiple such users at a time. Loftier optimization heuristics allow cost reduction in terms of processing power required, and leverage the best out of the current hardware architectures available. To this end, it is imperative to employ modern advancements in Machine Intelligence for the same, and hence papers dealing with such developments have been discussed as well. Cache optimizations, Auto-tuning, Instruction Level Parallelism, Feature Mapping, etc. form the basis of cutting edge technology, being extended to various consumer fields. This paper sheds a light on such heuristics and how they have been employed to the area of compiler design and optimization.**

***Keywords*: Compiler Optimization, Phase Ordering, Memory, Machine Learning, Survey**

1. Introduction

Today, parallelization is of utmost importance, considering the rise of multicore processors [6]. Currently, there are 2 ways of obtaining parallelization:

1. User level code
2. Compiler-generated optimized code

Of these 2, user-level code has been shown to be slightly effective at best, if the compiler generated code is not efficient. So it becomes necessary to come up with a good compiler level parallelization algorithm which can augment the user code for better parallel performance. Making sure that the resources available in a given system is utilized to its full capacity is one of the primary goals for all system programs. Another important aspect is proper optimization of the code.[7]

2. Literature survey

Jay Patel et al. in their research article, have surveyed numerous code optimization techniques. Their proposed system consists of making use of Artificial Neural Networks(ANNs) in the ordering of these optimization techniques. They have made use of the 4Cast-XL integrated into Jikes RVM optimization driver. The method involves the generation of a feature vector, profiles for the program, and then the use of the ANN, for every dynamically compiled method. ANNs have been used to predict the best optimization as well [1].

L Almagor et al. in their paper, aim to build an experimental study for large sequences and compute the accuracy against the benchmark models. Cost effectiveness of such models has been computed by studying the space of compilation sequences. Their prototype compiler build has a feedback loop and a steering algorithm to pick a compilation order, measure its impact, and adjust the compilation order. The datasets thus generated have been analyzed to gain insights into the structure of the spaces. Genetic Algorithms have been used and improved to make a better model fit [2].

Iterative Compiler Optimization has been shown to outperform static approaches, although it is a space expensive process. F Agakov et al. in their research, have developed a new methodology for the reduction of this complexity, and thus, speed up the optimization process. They have made use of predictive modelling to search the areas likely to give the greatest performance. The main approaches used here were the independent model and the Markov model, while evaluations were done on Texas Instrument C6713 and 1.27 on the AMD Au1500 platforms [3].

Sameer Kulkarni et al. in their work, have developed an approach to automatically select good optimization orderings on a per method basis within a dynamic compiler. A Markov process has been used for the characterization of the current state of the optimized code to create a better solution for phase ordering. The paper proposes to use a machine learning based approach which automatically learns a good heuristic for phase ordering. The second approach entails the use of Neuro-Evolution for Augmenting Topologies (NEAT) for the generation of customized optimization orderings for each method in the program. The NEAT has been used to construct phase ordering, and has been set to 60 neural networks in each generation, of which only 10 were propagated to the next generation. This process was allowed to continue and each successive generation of neural networks produces networks that perform better than the networks from the previous generation [4].

Out of the multitude of compiler optimizations available, Paul B. Schnek, in his work, has classified them into three categories: Machine Dependent, Architecture Dependent, and Architecture Independent. Machine Dependent and Architecture Dependent optimizations consider the structure of the computer, but not its detailed instruction set. Architecture Independent Optimizations are also global, but are based on the analysis of the program flow graph and the dependencies among statements of the source program. The McKeeman evaluation in Machine Dependent optimizations proposes a postprocessing technique for optimization, which is essentially a window traversing a sequence of unoptimized code. The classification of Architecture Dependent optimization is available for systems with n accumulators, or systems that can execute several instructions in parallel, or systems that execute multiple arithmetic and logical instructions upon multiple data streams. Architecture Independent optimizations have been split into three types: the frequency analysis, the matrix analysis, and the graph theoretic analysis [5].

Tarun Kumar [8] in paper aims to analyze various optimization techniques available for the compiler for a single core processor. Since different metrics produce a different type of optimization. Like a compiler may do space optimization or it may perform speed optimization. The type of optimization to be done depends on the application. GCC has 3 levels of optimization, O1, O2 and O3. Each successive levels applying more aggressive optimization techniques. Although it is not necessary that higher level will produce more optimized code.

To analyze optimization, the paper uses machine independent optimization techniques.

Multicube explorer is used for Design Space Exploration. Intel PIN was used for automatic feature extraction. MiBench was used for benchmarking and dataset generation for various application domains. The number of features was limited to 4 as an increase in features increases the search space exponentially. Graph of various optimizations done by O1, O2 and O3 is plotted over different application domains.

This paper presents an optimization method which uses both machine independent and machine dependent techniques for optimizing and transforming code for different machines. The goal is to produce a code which is optimized to work for a different system architecture but its code size is not > 10% of the original code. The steps given below are followed in sequential order:

- *Scalar Variables in Registers:* User-defined/local variables are put in registers to take advantage of spatial locality.
- Putting common expressions in registers: While the text code is scanned sequentially, a candidate table is constructed in which most common expressions are noted. Then these expressions are also loaded in the registers.
- *Using Special Registers:* So far, the techniques described were machine independent. In this step, using certain predefined information about the systems, we obtain the values of special registers. These registers specialize in certain tasks which makes it faster to use them for those tasks.
- *Pre-Optimization Transforms:* This is also a machine dependent task. Here we exploit the assembly level working of different machines to produce the optimal code. For eg, a system might allow its assembly language to load and store data from the registers in a single line. Using this, we can reduce the code size significantly.
- *Jump Optimization:* Some machines support short jump instruction, which allows the code to move forward or backwards from the current location of execution. A branch table is constructed and the distance of all the jumps are calculated. If the distance is high, the short jump is marked as long.
- After all these optimization techniques are applied, the final speed and space utilization are compared. The result obtained indicated the the optimization was successful.

3. Discussion

Given the number of projects requiring fast development of applications [9], it is necessary to find ways to optimize compilers in order to develop applications faster. There exists many methods to optimize compilers ranging from simple heuristics to complex Machine Learning applications. Each has its own strengths and limitations. There is no one-fits-all method that can be applied. It has to be a combination of all possible methods to optimize the compiler. In recent times, the rise of AI has given a unique opportunity in the way compilers can be optimized. It is true that optimizing compilers is difficult due to the varying hardware instructions

4. Conclusion

Compiler Design is a field of dynamic change with respect to sequence optimization and capitalization on instruction cache space. With cutting edge advancements in the fields of automation, Machine Intelligence, State Space Vectoring, etc., it is now possible to select a compiler that has both - a low resource utilization and a high throughput for available bandwidth. We have surveyed a few such papers to illustrate and highlight the current heuristics, and for researchers to draw conclusions from this to improve upon. To leverage the best out of the current available hardware with software optimization improves the architectural framework for any piece of application, and is bound to be an immensely useful tool for developers.

References

[1] Jay Patel, Mahesh Panchal - “Code Optimization in Compilers Using ANN”, International Journal of Computer Science and Mobile Computing, vol. 3, no. 5, pp. 557-561, May 2014.

[2] L. Almagor, Keith D. Cooper, Alexander Grosul, Timothy J. Harvey, Steven W. Reeves, Devika Subramanian, Linda Torczon, Todd Waterman, “Finding Effective Compilation Sequences”, Proceedings of the ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools, LCETS 2004.
[3] F. Agakov, E. Bonilla, J. Cavazos, B. Franke, G. Fursin, M.F.P. O’Boyle, J. Thomson, M. Toussaint, C.K.I. Williams - “Using Machine Learning to Focus Iterative Optimization”, School of Informatics, University of Edinburgh, UK, 2006.
[4] Sameer Kulkarni, John Cavazos, “Mitigating the Compiler Optimization Phase-Ordering Problem using Machine Learning”, Proceedings of the ACM International Conference on Object-oriented programming systems languages and applications, pp. 147-162, 2012.
[5] Paul B. Schneck, “A Survey of Compiler Optimization Techniques”, Proceedings of the ACM 1973 Annual Conference, pp. 106-113, 1973.
[6] K. S. McKinley, "A compiler optimization algorithm for shared-memory multiprocessors," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 8, pp. 769-787, Aug. 1998.
[7] H. S. Kim, M. J. Irwin, N. Vijaykrishnan, and M. Kandemir, “Effect of compiler optimizations on memory energy”, The Pennsylvania State University, 2000.
[8] T. Kumar and R. K. Singh, "Analysis of compiler optimization techniques by using feature mining technique," *2015 39th National Systems Conference (NSC)*, Noida, 2015, pp. 1-6.
[9] D. Boyle, P. Mundy and T. M. Spence, "Optimization and Code Generation in a Compiler for Several Machines," in *IBM Journal of Research and Development*, vol. 24, no. 6, pp. 677-683, Nov. 1980.
[10] Anjan Kumar Sarma, “New trends and Challenges in Source Code Optimization”, National Institute of Electronics and IT, 2015.
[11] Neeraj Kumar, Saroj Hiranwal, “Improving Code Efficiency by Code Optimizing Techniques”, IRJET, 2016.