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

A Brief Survey of Compiler Optimizations and their Sequence Optimization

Shiv Dutt Tripathi[1], Sashank Agarwal[2], Rohit Kumar[3], A. Parkavi[4]
[1,2,3]*Student, Dept. of Computer Science and Engineering, Ramaiah Institute of Technology, Bangalore, India*
[4]*Assistant Professor, Dept. of Computer Science and Engg., Ramaiah Institute of Technology, Bangalore, India*

***Abstract*: This paper is intended to provide insights about different optimizations available and their application details in compiler design. Optimization of compilers means tuning of objectives like latency, size, resource consumption etc. The most powerful thought about optimizations is that putting all of them at once does not solve all problems at once. This paper therefore also discusses different orders in which optimizations can be applied to have a much bigger impact with increased efficiency in desired sectors. Two compilers are also compared here.**

***Keywords*: Compiler optimization, Conflicting Pairs of Optimizations, Feature Space Exploration, Increased Efficiency, Optimization Technique Combinations, Reduced Latency.**

1. Introduction

In embedded world the efficiency of the software executed on the processor holds great importance. Specialized architectures pose a great deal of problems for good quality of code because of the old compiler technology and sometimes the incompatibility between instruction sets and the high-level languages. Re-targetable compilers were proposed for the same problem as when the designer feeds them with a well-versed account of the target architecture they can generate code for different architectures. This states that low-level optimizations can sometimes be let go of when targeting, but in such situation the efficiency drops. The paper talks more about how low-level optimizations can help increase efficiency of re-targetable compilers.

Wireless Sensor Networks have memory restrictions, program storage restrictions etc. They also need good optimization techniques on these ends to give better results. The combination of optimizations that do lead to better WSN performance are discussed in the paper. Since the compiler optimizations are endless what optimizations should we use in what order and their efficiency derived through optimized sequence of optimizations is also discussed. The conflicts are taken in account when discussing the optimal sequence of optimizations. Previous works targeting the problem used search techniques of some sort to cope up with the problem but the search space is huge and exponential which makes the task unprofessional. The paper discusses the algorithm that makes the solution to the problem a mere fully formalized set of steps. The code needs many transformations to meet the architecture demands. The optimizations can be used in handy for the same process.

Now - O0, -O1, -O2 and -O3 are the four definitions GCC has given for these optimization techniques.. These have complex relationships among them and using a bulk of them will not give the optimal result. The concept of feature mining is described in the paper. A program execution spends most of it's time in a small region of code [14]. The '90-10 rule' says that 90% of execution time comes from 10% of code [14]. Intrinsically 85% from loops and 15% from function calling – often called hotspots. Thus study of feature space gives one room for study of particular optimizations and their advantages on the specific cases.

2. Related work

A. *Literature survey*

The GCC compiler is said to have a pipeline description, which may/may not be used for RISC and VLIW processors [1]. The description language used in GCC is not so powerful as it does not cover the instruction flags. The fundamental aim of WSN is to collect statistical data and pose control over it's environment [2]-[4]. The organizability of WSNs holds the key to their effectiveness, they can be deployed over any region of importance and they must be able to organize themselves as a wireless network [5]-[7]. Therefore, a WSN is a self-organizing wireless network with the ability of producing, processing and transmitting application data. The durability expected may range from days to years [8].

The most important aspect of compiler optimizations is the order and combination part of it. For specific parts of the code these combinations of optimizations might conflict and results in one optimization which is used to diminish the performance enhancement achieved by any other. The conflicts so obtained are mainly from loops, function call boundaries, conditional statements etc. [9]-[13]. These conflicting pairs are called optimizations conflicting pairs (CPs). The discussion in this paper elaborates the idea for an algorithm to solve the conflicting pairs problem and do so in a formal paradigm. The optimizations that will be obtained after applying the algorithm will be Optimal Sequence of Optimizations (OSOs) [13]. Previous work in the area of providing with a set of optimal

optimizations depended on the idea of iterative compilation. The techniques used were Local minima search techniques, Pruning strategy, Genetic algorithms, Geometrical transformations, Pareto optimal. The machine learning approach is also used in way to extract static and dynamic features from the code and then use a Design of Experiments (DOE) [14] containing tool to search and find out the optimal optimizations to apply. This technique is explained in the paper along with the tools necessary to come up with results.

B. Comparison

Let us see two compilers known as GCC, LLVM which works on the EISC processor [15]. When compared both of them, we can see that LLVM is having calculation optimization better but in case of register allocation and jump optimization, GCC is excellent when compared to LLVM. Overall, the GCC compiler has better performance about 18% on average in terms of dynamic instruction. In addition, the compiled code size by GCC is smaller than that of LLVM. Table 1 shows comparisons of both architecture on the basis of instruction set.

Table 1. Comparison

	RISC	CICS	EISC
ISA Structure	Simple	Complex	Mode -rate
Program code Size	Large	Small	Mode -rate
Weakness	Difficult to develop 16 bits architecture	Complex in implement-tation	

3. Discussion

1) Pipeline aspects

A very detailed pipeline description is needed for re-targetable compilers. The forwarding nature is present in many processors which poses problems. The feature is essential as it also overcome many pipeline hazards. Also, exclusive write is a property in some processors: if there are two instructions which are consecutive and both of them write the same operand and if the order of execution makes second to execute over first then too the order of writes must be preserved. An instruction's behavior can be described in many ways on the pipeline, of them two can easily be discussed. The first description can be viewed as when assuming the description of each instruction, the stage's description holds good for the same. The second description uses the fact that instructions on the basis of read/written operands can easily be grouped. So they are grouped each group now is associated a pipeline pattern. The model explained has the following structure: stages section, pipeline section, value holding section, auxiliary resource section [1].

2) Pipeline Model

- *Stage Section:* In order to get the best of a pipeline structure there needs to be a set of stages, each of them performing the task individually and passing the results to the next stage. This promotes parallel architecture and provides of efficiency. The first line of this section has a tabular description subdivided into processor's pipeline stages and for each of them, their stage description follows.
- *Pipeline pattern section:* Before entering this stage the instruction set is divided into subsets. A pipeline pattern groups similar sections.
- *Value holding resource section:* The description for all the value-holding resources is provided in this section. Memory, registers and forwarding network are the value holding resources.
- *Auxiliary resource section:* Resources such as functional units need their description to be provided in this section.

3) Model based instruction scheduling results

The model based instruction scheduling results in observable speed optimization in VLIW or VLES based target architectures. The core idea is of reordering the code of the application which in turn minimizes latencies.

4) WSN optimizations

Putting all 150 optimizations to WSN available on GCC is impractical. Some important optimizations, which affect the code size considerably, are chosen over others and put to test. Total 61 optimizations are chosen but the feasibility is still high so they are divided into 15 groups, which can be tested. Both the code size and the performance are conflicting in some instances when the results are compared. Since sometimes the optimization techniques use some extra code space for better control flow constructs. The basic trade here is of code space in order to get better performing code. The energy side of the code is highly marketed.

5) Optimized sequence of optimizations

Finding the optimized sequence of optimizations using the graphical methodology of choosing three triangles of optimizations and out of all triplets the optimized ones to be applied are chosen. The sequences are graded on their performance as triplets and the resultant grades are the basis for choosing the optimized sequence. The process is compared to traditional approaches and the results report an improvement of 41% in some cases. Since the proposition of optimized sequence of optimizations is very important to be worked on, the triplet method is an important study to be included.

6) Feature mining technique

This is another approach to designing a space explorer, which will observe behavior of different optimizations techniques on different source codes and divides the static and dynamic features. This method is still not combined with the above discussed method of OSOs. The analysis of compiler optimizations can be easily done using the feature mining technique and using tools like the M3 Explorer and PIN tool. The explorer provides with features like automatic design space exploration, portability and modular composition. The feature extraction part is done using the Intel PIN based dynamic profiling framework. This tool extracts the two kinds of features

there are i.e. static and dynamic. The tool will however give results based on the architecture. The point to note from the results of this analysis is that when more number of optimizations are applied the number of movements by the program counter decreases. Stack reads and writes in turn are reduced thereby stating there is less need of storing intermediate values. This in turn makes executables faster.

7) Loop-invariant optimization

Compiler optimization and its consequences on real-time systems is explained where control flow in code is generated by an optimizing compiler is generally different from the control flow generated by user's source code. Therefore, this might result in setting breakpoints at incorrect locations in the code or it might fail to set relevant breakpoints. In addition, these scenarios are being given under the Debugging Highly Optimized Ada with Code Motion (DHACM) program. The goal of this program is to improve the performance of the debugger when applied on optimized code.

8) COS and COSP

The compiler optimization set (COS) and Compiler optimization selection problem (COSP) are discussed about the problem of using the same COS in a fixed order to all programs, it can improve the performance of some benchmarks while degrading the performance of others. Further architecture Fig. 1, where exploratory space has some performance counter set and 'N' COS for many programs. In addition, COSP Mitigate is used to identify the best COSs of exploratory space. In addition, the compiler optimization sequence validator is used for the validation and verification for which COS will be the best one.

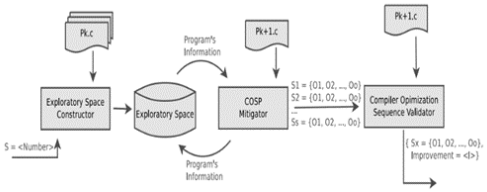

Fig. 1. The architecture

4. Conclusion

The pipeline model is discussed as to study how the pipeline models can affect optimizations. The most important part of upcoming communication networks are WSNs and the need for code optimizations is high inside a WSN. The discussion about WSN code optimizations aims at putting more effort in making these networks more efficient. The optimizations cannot be just put in any order and so the Optimized Sequence of Optimizations using the triplet method is discussed. The feature mining technique focusses on the use of two tools and the technique can be used along with OSO to come up with better set of optimizations.

References

[1] Ghica, Lavinia, "An accurate pipeline model for optimizing retargetable compiler", 9th International Conference on Intelligent Computer Communication and Processing (ICCP), pp. 283-286. IEEE, 2013.
[2] I. F. Akyildiz, "Wireless Sensor Networks", John Wiley & Sons Inc, 2010.
[3] A. Hac, "Wireless Sensor Network Designs", San Francisco, CA, USA: John Wiley, 2003.
[4] M. Ilyas, "Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems", CRC PRESS, 2004.
[5] A. L. L. Aquino, "Wireless Sensor Networks for Monitoring Amphibians," in Proceedings of Grand Challenges in Computer Science Research in Latin America Workshop, 2008, pp. 25–49.
[6] R. Herlien, "An Ocean Observatory Sensor Network Application," in Proceedings of IEEE Sensors Conference, 2010, pp. 1837–1842.
[7] C. Goumopoulos, "Ambient Ecologies in Smart Homes", Comput. J., vol. 52, pp. 922–937, November 2009.
[8] L. B. Ruiz, "On the Design of a Self-managed Wireless Sensor Network", IEEE Communications Magazine, vol. 43, no. 8, pp. 95–102, 2005.
[9] C. Norris, "An experimental study of several cooperative register allocation and instruction scheduling strategies", 28th annual international symposium on Microarchitecture, pages 169–179. IEEE Computer Society Press, 1995.
[10] D. Berson, "Integrated instruction scheduling and register allocation techniques", Languages and Compilers for Parallel Computing, pages 247–262, 1999.
[11] R. Kumar, "Compiling for instruction cache performance on a multithreaded architecture", 35th annual ACM/IEEE international symposium on Microarchitecture, pages 419–429. IEEE Computer Society Press, 2002.
[12] M.E. Wolf, "Combining loop transformations considering caches and scheduling", 29th annual ACM/IEEE international symposium on Microarchitecture, pages 274–286. IEEE Computer Society, 1996.
[13] Asher, "A study of conflicting pairs of compiler optimizations", 11th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), pp. 52-58. IEEE, 2017.
[14] Kumar, Tarun, "Analysis of compiler optimization techniques by using feature mining technique", 39th National Systems Conference (NSC), pp. 1-6. IEEE, 2015.
[15] Chanhyun, Miseon Han, "Performance comparison of GCC and LLVM on the EISC processor", International Conference on Electronics, Information and Communications (ICEIC), pp. 1-2. IEEE, 2014.