ISSN No. 0976-5697

Volume 8, No. 5, May-June 2017

International Journal of Advanced Research in Computer Science

REVIEW ARTICLE

Available Online at www.ijarcs.info

A Comprehensive Study on Code Optimization- Levels & Techniques

Neha Bhateja
Department of Computer Science & Engineering
Amity University Haryana, India

Abstract: Language translators are required to convert one language into another language. Compiler is one of the language translator which is used to convert the high-level language into machine understandable language. A good compiler can have a huge effect on code performance. for this purpose, code optimization is performed to improve the speed and size of the code.

Keywords: Compiler, Code Optimization, Code Generation, Dead code, Branch Optimization.

INTRODUCTION

Compiler is one of the software development tool which is used to convert the high level language into assembly language which is further converted into machine language in terms of 0's and 1's with the help of an assembler. The compiler performs its tasks into various phases. Each phase works on the output of previous phase. Following are the phases of a compiler structure:

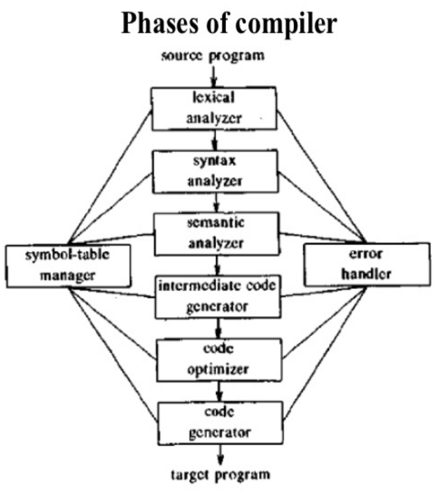

Fig: Phases of Compiler [2]

Lexical Analysis Phase: It accepts the character from the source code and group them into stream of tokens such as a keyword, an identifier, a character. The sequence of characters forming a token called lexeme.
Syntax Analysis Phase: Syntax Analysis phase accepts the output of lexical phase as an input. It generates a structure on the token streams. This structure is known as parse tree.
Semantic Analysis Phase: the important component of semantic analysis is type checking. It checks the source code for semantic errors and missing information.
Intermediate Code Generation: the syntax and semantic analysis phase generate an intermediate code of the source code which is easy to produce and translate into machine code. Intermediate representation can be done in various forms like three address code, quadruples.

Code Optimization: to improve the performance in terms of speed, the code optimization is performed on the intermediate code.
Code Generation: Code generator phase is responsible to generate the final code i.e. machine code.

CODE OPTIMIZATION

Code optimization is a technique to transform the source code by making some changes in the code with an intent to achieve the quality and its efficiency.
Code optimization is performed by reducing the length of the code by removing the repeatable and unnecessary lines of code. An optimized program becomes smaller in size, used less system resources like memory and rapidly executes all input and output operations [1].
During compilation process, optimization can be performed at various phases.

- Beginning Phase: At this phase user can use better algorithms to alter or rearrange the code.
- Intermediate Phase: Compiler can improve loops, procedure calls, and address calculations
- Target Phase: While producing the target machine code, the compiler can use the CPU registers efficiently.

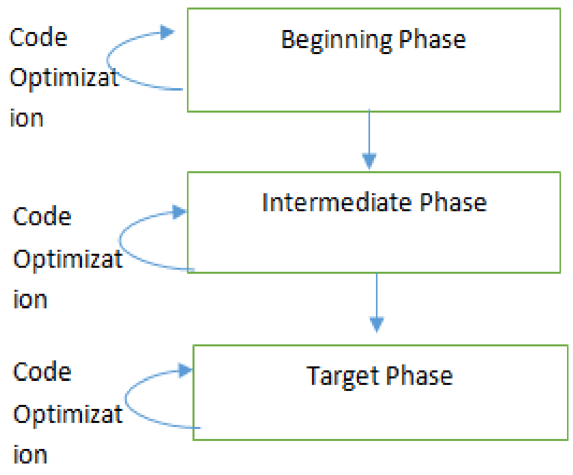

Fig: optimization at different phase

BENEFITS OF OPTIMIZATION PROCESS

To perform the optimization on code helps the program developers to achieve their goals like [4]:

- Higher Performance

The overall performance of the system is degraded due to the maximum execution time taken by the application code to execute. By using the optimization process, the compiler produce code much faster which helps the developers to gain the better performance in term of execution speed.

- Lower Development Cost

By opting the optimization techniques system development cost can be reduced. For example, a better performance program could be executing without using a higher cost processor. Memory requirements can be decreased by optimizing the code which further helps the developer to reduce the production cost by using less memory.

- Less Development Time

The development of software applications becomes more complicated day by day. As a result, the concept of reuse the code is one of the best method which helps the developers to reduce the development time and to deliver the new products on time.

LEVELS AND TECHNIQUES OF OPTIMIZATION PROCESS

Optimization process can be performed either manually by the programmers or using the automatic optimizers, called optimizing compilers [5].

Optimization process is categorized into two levels i.e. High level and Low level optimization process. High level optimization process is performed with a goal to optimize the design of a structure. It is performed by the programmers who can handle functions, classes, loops, branches in source code etc. whereas, low level optimization is performed when source code is compiled into the machine code. Following are the code optimization techniques [4,5]:

- Branch Optimization: the source code is rearranged in such a way to reduce the logic of branches and to merge the separated blocks of program code.
- Common subexpression elimination: In this, those expressions or operations that represent the same meaning are eliminated by using the previous value.
- Dead Code elimination: A part of the source code that never be accessed or its value never be used in the program should be eliminated.
- Dead store Elimination: Dead Store elimination technique helps the compiler to eliminate those variables who stores a value but never be used. These variables are called dead store, which waste the system resources like memory and processor time.
- Constant Propagation: constant propagation means assigning the value of a constant to the variables at the compile time. It can also simplify the conditional branches to one or many unconditional statements.
- Global Register Allocation: Register allocation is a process of assigning the CPU registers to the variables. Allocation can be done at various levels like, local allocation, global allocation and inter procedural allocation. The compiler used the concept of graph coloring algorithms at the time of compilation to allocate the number of variables to few set of registers.
- Instruction Scheduling: As, all the instruction defined in the source code are essential. To maximize the instruction parallelism, instruction scheduling technique is used in compiler optimization process. In this, the instructions are rearranged or rewrite to improve the performance of the system in term of execution time without changing the actual meaning of the code. It is performed before or after the register allocation process.
- Interprocedural Analysis: this is one of the code optimization technique that perform the analysis on whole source code rather than a single block. It is used to eliminate the duplicate calculations and those expressions that used inefficient memory.

ISSUES WITH THE OPTIMIZATION PROCESS

- The basic idea of optimization process is to improve the overall performance of the system rather than improving the complexity of an algorithm [3].
- A compiler handles only those set of instructions that are grouped together in a single module or file. It cannot be considered the referenced information that is defined or locate in another module.
- While dealing with the huge program code, the optimization process is a time consuming as it takes a lot of time.
- Due to increase the complexity between the interaction of different modules, it is impossible to find the optimal result from the optimization process

CONCLUSION

Code optimization is used to optimize the source code. The developers can achieve their targets in term of execution speed, cost and reduce development time by using the various optimization techniques. Optimization process helps the compiler to generate an optimal machine code. The performance of a compiler is one of the important factor which affect the performance of a program. So the developers have to check the optimization capability before selecting any compiler.

REFERENCES

1. Clinton F. Goss (August 2013) [First published June 1986]. "Machine Code Optimization - Improving Executable Object Code" (PDF) (Ph.D. dissertation). Computer Science Department Technical Report #246. Courant Institute, New York University. arXiv:1308.4815 Retrieved 22 Aug 2013.
2. Cooper, Keith D., and Torczon, Linda, Engineering a Compiler, Morgan Kaufmann, 2004, ISBN 1-55860-699-8 page 404
3. T. Jones, M. O'Boyle, and O. Ergin. Evaluating the effects of compiler optimisations on AVF. In INTERACT '08: Proceedings of the 2008 Annual Workshop on the Interaction T. Jones, M. O'Boyle, and O. Ergin. Evaluating the effects of compiler optimisations on AVF. In INTERACT '08: Proceedings of the 2008 Annual Workshop on the Interaction.
4. Chabbi, M.M., Mellor-Crummey, J.M., Cooper, K.D.: Efficiently exploring compiler optimization sequences with pairwise pruning. In: Proceedings of the 1st International Workshop on Adaptive Self-Tuning Computing Systems for the Exaflop Era, EXADAPT 2011, pp. 34–45. ACM, New York (2011)

5. Epshteyn, A., Garzarán, M.J., DeJong, G., Padua, D.A., Ren, G., Li, X., Yotov, K., Pingali, K.K.: Analytic Models and Empirical Search: A Hybrid Approach to Code Optimization. In: Ayguadé, E., Baumgartner, G., Ramanujam, J., Sadayappan, P. (eds.) LCPC 2005. LNCS, vol. 4339, pp. 259–273. Springer, Heidelberg (2006).