ISSN: 0975-766X
CODEN: IJPTFI
Research Article

Available Online through
www.ijptonline.com

IMPROVING OPTIMIZATION SEQUENCE OF COMPILERS BY USING SEQUENCE SELECTION APPROACH

Ramani S*[a], Boominathan P [a], Swathi J, Narayanan[a], Rajkumar S[a]
[a]School of Computing Science and Engineering, VIT University, Vellore-14.
Email: ramani.s@vit.ac.in

Received on 15-05-2016 *Accepted on 18-06-2016*

Abstract:

Compilers have many optimization sequences to be run on a program IR (Intermediate Representation). Applying all optimization sequence leads to degradation of performance. It needs a best optimization set from optimization sequence for every program to improve performance. The literature addresses standard universal optimization sequences set derived from set of program spaces. Retrieving best optimization sequences form this set consumes more time. This paper proposes a sequence selection approach that reduces time for selecting best optimal sequence set and further it reduces runtime of a program space. The experimental results showed improved performance on different program spaces in different bench mark suites.

Keywords: Optimal Sequence, Benchmark Suites, Intermediate Representation

Introduction

Compiler is the main component for any programming language which is used for converting the one language to another language. Compilation of any program has 6 phases [24]. The compiler generates Intermediate representation (IR) for a program when it is processed through the syntactic and semantic analysis [3]. The next phase of the compiler is code optimization which is used to optimize the IR code with respect to parameters such as memory, run time and power consumption [2].

The optimization phase which consists of many optimization sequences which are helpful for optimization of IR code [8]. The optimized code is given as input for the next phase of the compiler for generating the code for target system [10]. The execution time and memory of a program depends on the optimization phase.

Related Work

John Canvazoset.al [1] has proposed a best machine learning approach using hardware performance counters [2] to automatically select the best optimization options. In recent research, learned-models [4, 5, 6, 7] are useful for knowing the optimization sequences of the compiler.

Dynamic compilers can select best "optimizations use logistic regression technique [8]. The optimization sequence space size grows exponentially as a function of sequence length. If there are k optimizations, then there are k^loptimization sequences of length l [7, 8]. An optimization sequence is optimal for a program if it hints to least program runtime when associated to all other sequences [7, 6]. This definition is incomplete as the program runtime and its behavior could be a function of the input [8, 9]. An optimal sequence for a program depends on the program characteristics as defined by the input to the program [12, 13]. A program with different input distribution and parameters with respect to that input distribution almost remains constant [11].

Predictive heuristic technique is an approach to find optimization sequences. Unpredictable interaction among optimizations in this technique leads to performance degradation. Spyridon et al [19] proposed optimization sequence exploration technique [20, 21, 22, 23] (OSE) as first iterative compilation model for general purpose compilers to overcome the problem in predictive heuristics.

Iterative compilation techniques [15] use efficient algorithms to find good optimization sequences. In this technique program evaluations are very large and not feasible to apply in applications. In machine learning approaches [17] prediction models are used to determine good optimization sequences. Prediction models learn by running and observing some set of programs and their runtimes. It will take decisions on optimizations to be applied on program space [18].

Thomson.et.al [16] proposed a clustering approach to cluster training programs using feature vectors. The best optimization sequence for a new program space lies at the cluster centroid [13, 14]. Park.et.al proposed a technique to select optimization technique to select optimization sequence selection using tournament predictors [16].Suresh purini et.al[2]used three different benchmark suites are to find best optimization sets.

These techniques used LLVM(low level virtual machine) test suite[14] where 61 optimization sequences are present. These programs are also called as microkernel programs. Microkernel programs have lesser execution times and consist of programs like searching and floating Point arithmetic. Iterative compilation techniques [13] are applied to find near optimal optimization sequence sets. Finally testing the effectiveness of this approach using Mibench benchmark suite [12] and polybench benchmark suite programs shows an improvement in speedup.

Prior Work

In the existing system compiler optimization phase has default optimization sequences [2]. The sequences are applied on a program in sequential manner which optimizes different sections of the program. These sequences may wrongly communicate with each other which may affect the speed up of a program.

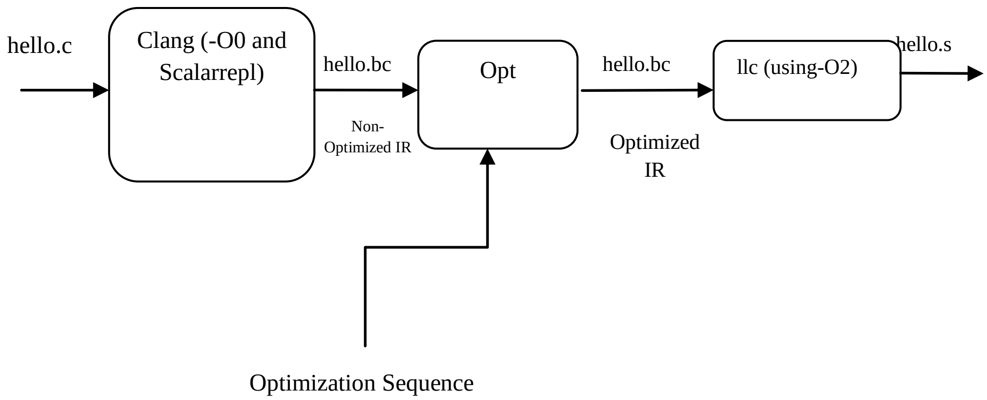

Fig1: Program compilation setup in LLVM.

LLVM [14] (low level virtual machine) compiler generates machine independent code. Clang [14] is one of the front-ends supported by llvm.Fig-2 describes the execution sequence of the program .Clang runs the programs and produces non-optimized Intermediate representation.

In the next phase of compilation clang run default optimization sequences like O1, O2 and O3 on the program to produce optimized IR. Finally optimized IR converted to machine code as bit code file.

Proposed Work

In the proposed System selection of best optimization sequences are done for poly-bench benchmark suite programs. Best set of optimization sequences are selected based on different program classes.

It uses the sequence selection algorithm to reduce the optimization sequence set so, that for every program class there exist at least one optimization sequence set.

By using the clustering algorithm it groups the similar sequences set. So, this approach reduces the execution time of a program by select the best optimization sequence thus increases the speedup of a program.

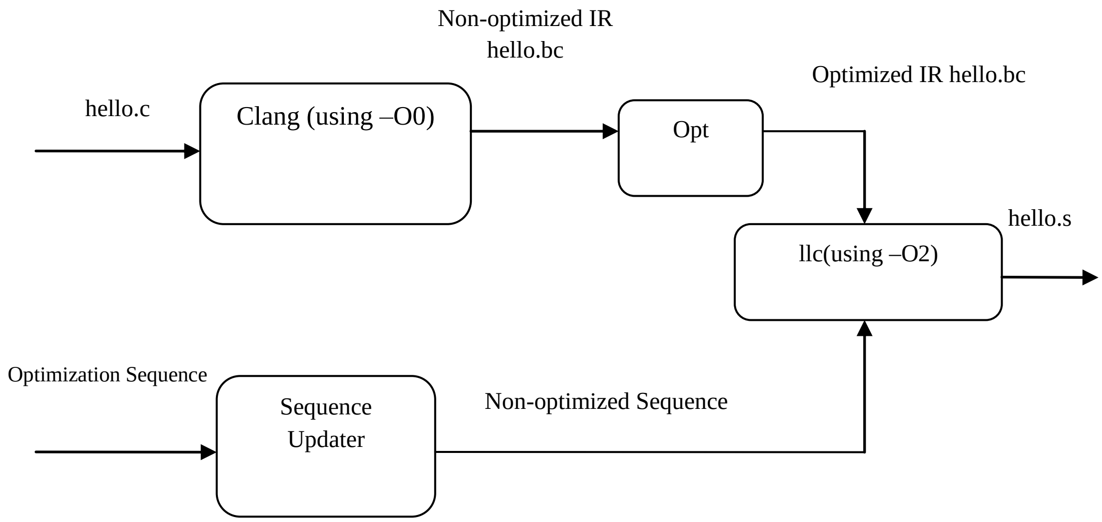

Fig2: Program compilation with updated optimization sequences.

Above figure describes the proposed approach using LLVM compiler execution sequence. Optimized IR produced by clang using default optimization sequences wrongly interact with each other. These kinds of interactions cannot give efficient runtimes to programs. Sequence reduction algorithm is applied on the program space selects good optimization sequences. These optimization sequences are given as input to the optimization phase of the compiler to reduce runtime of the program.

Sequence Selection pseudo code

Input: program and optimization sequence

Output: Best optimization sequence

Method:

Begin

bseq=optimization sequence;

btime=program execution time using bseq;

v=1;

while v=1 do

v=0;

bseqlength=length of bseq;

for i=1 to bseqlength do

current sequence=remove the l^{th} optimization from the bseq;

current sequence time=execution time using of current sequence;

If current sequence time≤btime then

Update b time with current sequence time;

Update bseq with current sequence;

Change v to 1;

break from for loop

End of if

End of for

End of while

Return bseq.

End of main

Fig 3: Pseudocode for Sequence Selection

The above algorithm extracts best optimization sequences by giving the program and optimization sequence as input.

Experimental Analysis

Default optimization sequence(O1) and best optimization sequence(O2) applied on the poly benchmark program by using the VPO tool which gives the execution time for each program in the program space.

Result Analysis

Table 1.1 Runtime of default optimization sequence and best optimization sequence set over the poly benchmark suite.

Benchmark Programs	Default optimization sequence	Best optimization sequence set
correlative.c	0.0976	0.097
covarien.c	0.198	0.193
2mm.c	1.8	1.65
3mm.c	4.74	4.5
atax.c	3.46	3.2
cholesky.c	5.36	5.1
doitgen.c	2.15	1.5
gemm.c	2.65	2.35
gemvar.c	6.54	6.14
gesummv.c	4.34	3.29

The following graph illustrates the comparison between the default and best optimization sequence set runtime over the polybenchmark suite programs.

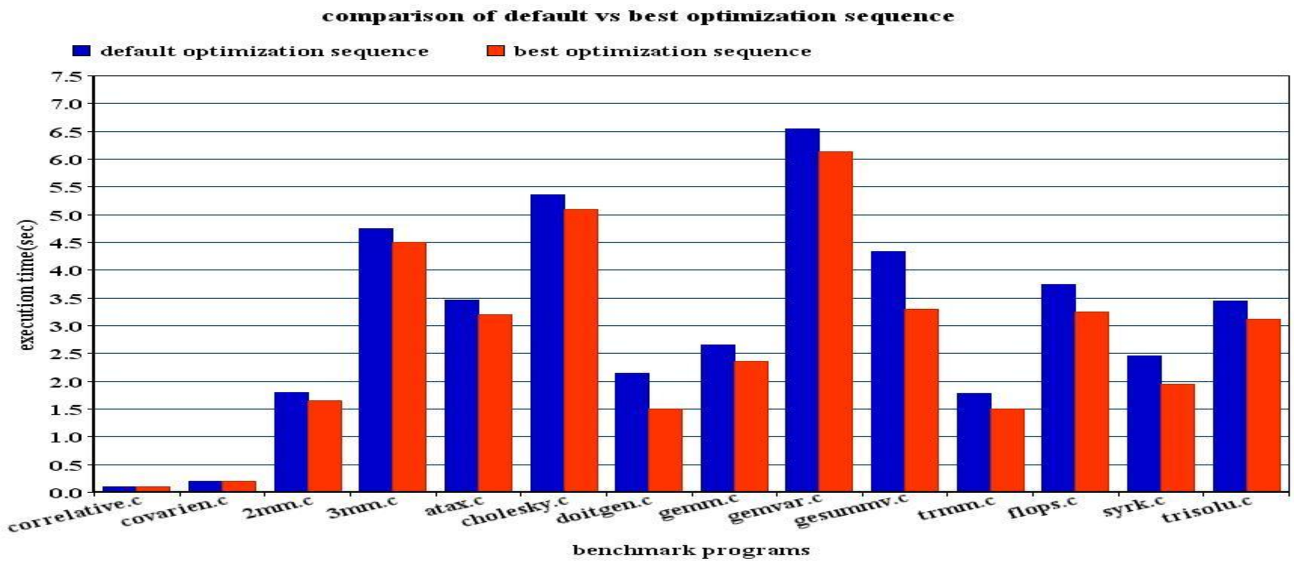

Fig 3.Graphical Representation of default and best optimization sequence sets execution timeover the polybenchmark programs.

The above graph illustrates the compilation time of each polybenchmark program using the default optimization sequence is less than the compilation time of each polybenchmark program using the best optimization sequence .From the graph we can say that best optimization sequence gives better performance than the default optimization sequence.

Table 1.2 Speedup for default and best optimization sequence of polybenchmark programs

Benchmark Programs	Speedup of Default Optimization Sequence	Speedup of best Optimization Sequence
correlative.c	1.006	1.006
covarien.c	1.009	1.025
2mm.c	0.74	1.09
3mm.c	0.89	1.053
atax.c	0.095	1.0812
cholesky.c	1	1.05
doitgen.c	1.35	1.433
gemm.c	1.015	1.1276
gemvar.c	1.019	1.065
gesummv.c	1.102	1.319

Table1.2 demonstrates the speed up of each benchmark program. Speed up of default optimization and best optimization sequence of each benchmark program has been plotted on the graph. Speed up is calculated as follows.

$$Speedup = \frac{defaultoptimization}{newoptmization} \quad (1)$$

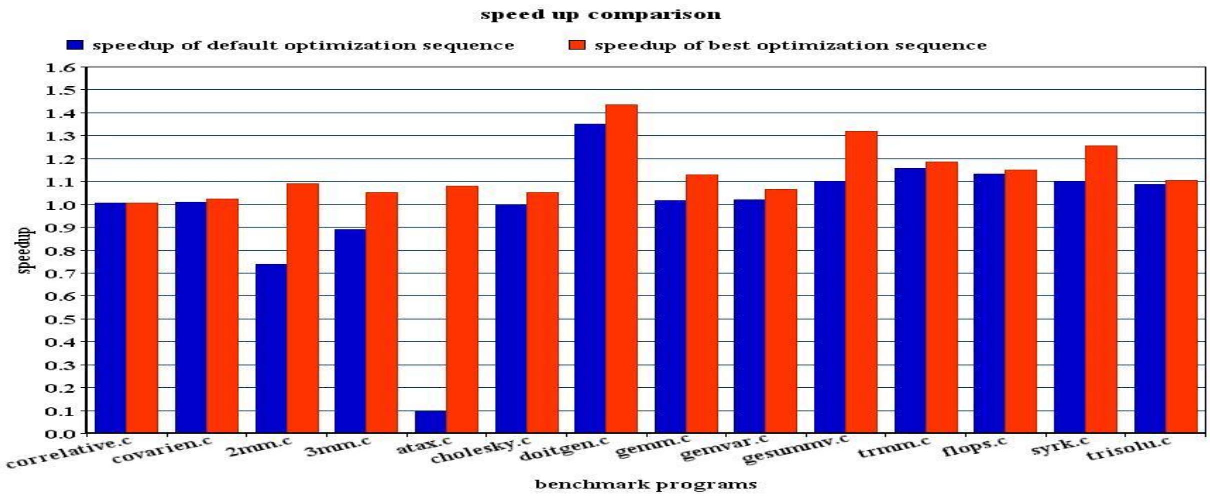

Fig 4: Speedup for default and best optimization sequence over polybenchmark programs.

The above graph illustrates the speedup of each benchmark programs using the default optimization sequence is less than the speedup of each benchmark programs using the best optimization sequence. From the graph we say that performance of the best optimization sequence is improved over the default optimization sequence.

Table1.3. Percentage improved for benchmark programs

Benchmark program	Percentage improved
correlative.c	6
covarien.c	2.5
2mm.c	9
3mm.c	5.3
atax.c	8.12
cholesky.c	5
doitgen.c	43.3
gemm.c	12.76
gemvar.c	6.5
gesummv.c	31.9

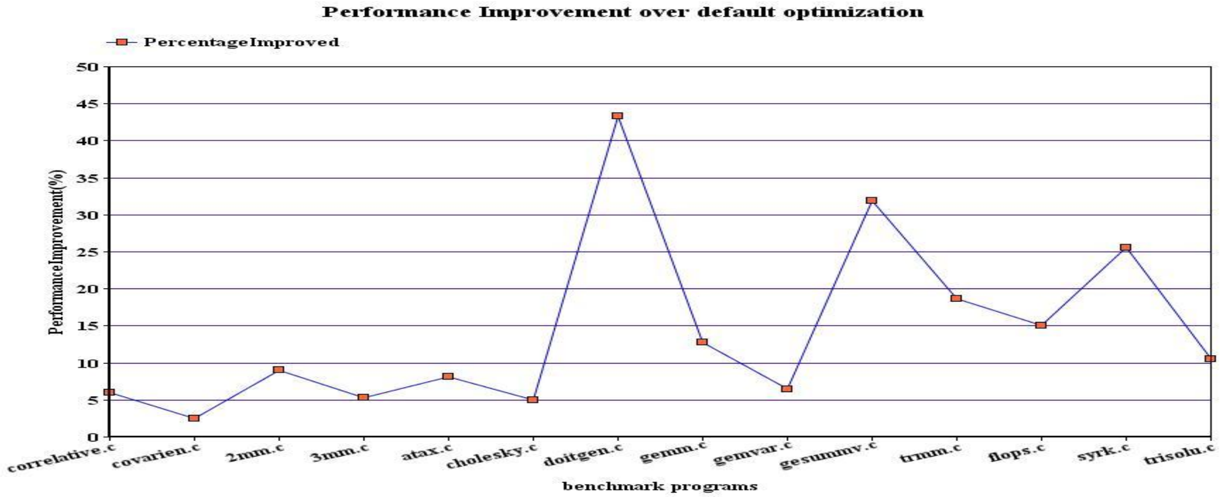

Fig 4: Performance improvement of best optimization sequence over the default optimization sequence.

$$\text{Percentage improved} = (\text{speedup} - 1) * 100 \qquad (2)$$

The above graph illustrates the performance improvement of benchmark program of best optimization sequence over the default optimization.

Conclusion:

In this paper we proposed a practical approach to solve the false interaction between optimization sequences. This method is quite different from the iterative compilation and machine learning-based prediction techniques. The idea is to filter the infinitely large optimization sequence on a program space using sequence selection algorithm. Selection of good optimization sequences set is done in very fast on a particular program space. Then given a new program we can try all the sequences from the good sequences set and choose the best sequence.

References:

1. Agakov,Bonilla,Cavazos,Franke,Fursin,Boyle,Thomson,Toussaint,Andwilliams,"Using machine learning to focus iterative optimization", In Proceedings of the International symposium on Code Generation and Optimization, Vol.15, no.7, pp.234-244, Sep. 2006.
2. Suresh purini and Lakshyajain, "Finding Good Optimization Sequences Covering Program Space", ACM Transactions on Architecture and Code Optimization, Vol. 9, No. 4,pp.63-68,Jan 2013.
3. W.Orchar-Hays, "The evolution of Programming System", ACM Transactions on Programming Languagesand Systems, vol. 19, pp. 1053–1084, Nov. 2009.

4. GrigoriFursin, John Cavazos,Michael o'boyle and Olivier Temam, "Creating The Conditions For A More Realistic Evaluation of Iterative Optimization", ACM Transactions on Iterative Optimization, Vol. 8, No. 3, pp.48-54,Sep. 2012.
5. AMD Compiler usage guidelines for 64-bit operating systems on AMD 64 platforms. http://www.amd.com/usen/assets/content type/white papers and tech docs/32035.pdf,2006.
6. Azimi, M. Stumm, and R. W. Wisniewski. "Online performance analysis by statistical sampling of microprocessor performance counters", ACM Transactions on Iterative Optimization, vol.6, No.7, pp. 101–110, Jan. 2005.
7. C. M. Bishop. "Neural Networks for Pattern Recognition". Oxford University Press, Oxford, UK, 2010.
8. E. Meijer and J. Gough, "Technical overview of the Common Language Runtime" [Online] Available: http://research.microsoft.com/~emeijer/Paper/CLR.pdf
9. J. L. Wilkes and Barthou "Application of microprogramming to medium scale computer design," In Proceedings of the International Symposium on Code Generation and Optimization Microprogramming, vol.5, No.4, pp.87-92, Feb 2008.
10. R. J. Hookway and M. A. Herdeg, "Digital FX! 32: "Combining emulation and binary translation", Dig. Tech. J., vol. 9, no. 1, pp.3–12, Jan 2007.
11. Chen, Huang, Eeckhout, Fursin, Peng, Temam, "Evaluating iterative optimization across 1000 datasets", In Proceedings of the 2010 ACM SIGPLAN conference on Programming Language Design and Implementation (PLDI '10). Vol.7, No.3, pp.448–4590, 2009.
12. M.R.Guthaus, Ringenberg, Ernst, Austin, "Mibench: A free, commercially representative embedded benchmark suite", In Proceedings of the IEEE International Workshop on Workload Characterization.
13. M.Boyle, "Finding near optimal sequence using iterative compilation technique", In Proceedings of the International Symposium on Code Generation and Optimization, Vol.5, No.8 pp. 275–320, 2009.
14. Lattner, Adve, "LLVM: A compilation framework for lifelong program analysis & transformation", In Proceedings of the International Symposium on Code Generation and Optimization: Feedback-Directed and Runtime Optimization (CGO '04).

15. Park, Kulkarni, and Cavazos, J. 2011, "An evaluation of different modeling techniques for iterative compilation", In Proceedings of the 14th international conference on Compilers, Architectures andSynthesis for Embedded Systems (CASES '11). ACM,Vol.5, No.8 pp. 65–74,2007.
16. Thomson, Oboyle, M.,Fursin, and Franke, B. 2009, "Reducing training time in a one-shot machine learning-based compiler", In Proceedings of the International Conference on Language and Compilers forParallel Computing. Vol.6,No.8,pp.399–407,2009.
17. Leather, Bonilla, and Boyle, M. 2009. "Automatic feature generation for machine learning based optimizing compilation", In Proceedings of the 7th Annual IEEE/ACM International Symposium onCode Generation and Optimization (CGO '09),Vol.11, No.4, pp.81–91, 2009.
18. Michellehribar, Haoqiangjin, Abdulwaheed and Jerryyan 2009, "A machine learning-based compiler", In Proceedings of the International Conference on Language and Compilers forParallel Computing, Vol.9, No.7, pp.399–407, 2008.
19. SpyridonTriantafyllis. Manish Vachharajani. Neil Vachharajani. David I. August,"Compiler Optimization-Space Exploration", In IEEE '09 Proceedings of the International Symposium on Code Generation and Optimization.Vol.8, No.15, pp.340-346,2007.
20. T. Kisuki, P. M. W. Knijnenburg,Boyle, F. Bodin, And H. A.G.Wijshoff, "A feasibility study in iterative compilation," in InternationalSymposium on High Performance Computing, Vol.7,No.9,pp.121-132,2010.
21. F. Bodin, T. Kisuki, P.M. W. Knijnenbsurg, M. F. P.O.Boyle, and E. Rohou, "Iterative compilation in a non-linear optimization space," In Proceedingsof the Workshop on Profile and Feedback-Directed Compilation ,in Conjunction with the International Conference on Parallel Architecturesand Compilation Techniques,Vol.4,No.8,pp.78-89,October 2011.
22. D. L.Whitfield and M. L.Soffa, "An approach for exploring code improving transformations," ACM Transactions on Programming Languagesand Systems, vol. 19, pp. 1053–1084, November 2011.
23. Alfred V.Aho, Monica S.Lam, Ravi Sethi, and Jeffrey D.Ullman, "Compiler Principles, Techniques, and Tools", Pearson Education India, Second Edition.

Corresponding Author:

Ramani S*,

Email: *ramani.s@vit.ac.in*