Turkish Journal of Computer and Mathematics Education *Vol.12 No.6 (2021), 3913-3922*

Research Article

A Content Based Filtering Approach for the Automatic Tuning of Compiler Optimizations

Rafeef M. Al Baity [1,1] , Esraa H. Alwan [1], Ahmed B. M. Fanfakh [1]

[1] Department of Computer science Collage of science for women University of Babylon

[1]rafeefm.albayati@gmail.com, [2]esraa.hadi@uobabylon.edu.iq, [3]ahmed.fanfakh@uobabylon.edu.iq

Article History: Received: 10 November 2020; Revised 12 January 2021 Accepted: 27 January 2021; Published online: 5 April 2021

Abstract: Recently a large number of compiler conversions have been implemented to optimize programs. A comprehensive exploration of all possible sequences of optimization is not practical because the search space is huge considering the large number of compiler optimizations passes. In addition, predicting the effectiveness of these optimizations is not an easy task. In this work, the suggested approach offers automatic tuning of compiler optimization sequences in place of manually tuning by recommended optimization sequences based on program features. Techniques inspired from the Recommendation System (RS) field to provide a solution to the autotuning of compiler optimizations problem. Content Based filtering method is finding a group of programs that are closest to the unseen program based on the similarity of their features. Then the best optimization sequences for these programs are recommended to the unseen one. Two versions of the CBF method, with and without rate value are presented.

The approach is evaluated using three benchmark suites PolyBench, Shootout, and Stanford, including 50 different programs and using LLVM (Low Level Virtual Machine) compiler passes down Linux Ubuntu. Results obtained showed that such method is superior to the standard level of optimization -O3 of LLVM compiler in improving the execution time by an average of 9.3 % for CBF without rate, 13.7% for CBF with rate.

Key words: Compiler**,** Pass, Optimization Sequences, Recommendation system

1 Introduction

Compilers consist of three stages: the front-end, middle-end and back-end. The front end analyzes a source program and the creates an intermediate representation (IR), the middle end represents the optimizer, applies a series of transformations to the IR to get best execution time, size of the code, and energy consumption, the back-end takes IR and outputs the target language. Compilers perform their optimizations in passes, where each pass is responsible for a specific code transformation. Compiler has many optimizations passes to be run on a program IR (Intermediate Representation). The optimization of the second stage (optimization) plays an importance role for the performance metrics. In other words, enable compiler optimization passes (e.g., loop unroll, allocation register , etc.) may yield great benefits in several the performance metrics where these performance metrics could be code size, the execution time or power consume[1].

compiler typically contain some standard optimization levels that automatically enables the user to include a set of pre-defined sequences optimization of the assembly process [2,3]. It is known that these standard improvements (eg -O1 or -O2 or -O3 or -O) useful for performance (or the size of the code) in most cases. In fact, powerful optimizations can weaken the code they are applied to. Therefore, it is difficult to determine whether to enabled specifically compiler optimizations passes on the target code [2]. Thus, to resolve these problems, this worked presents a compiler autotuning method derived from the Recommendation system. The Content Based Filtering method, which uses performance counters to characterize program is introduce to find the best optimize sequences in terms of execution time.

In this approach, we find optimizations sequences which performed well on a "similar" (previously explored) program will work well for the new program being compiled. Using performance counter events is attractive as it exploits knowledge of the program's dynamic behavior. This paper is organized as follows: the next section (2), surveys some related work, then section (3), gives details of the proposed method. Section (4) presents some experimental results from the implementation of the Content Based

Filtering method with and without rating value. Finally, in section (5), conclusions for the proposed method are presented.

2 Related Work

In this section, it is listed many of the related work that has been done in this area. Manal H. Al Mohammed 2020 [4], In order to obtain best optimization sequences that can improve program performance, a parallel genetic algorithm approach has been proposed. In this method, the programs were classified into three groups and then three copies of the genetic algorithms were applied, each one to be grouped in parallel. Where three optimal improvement sequences were obtained. When the results of the implementation time compare to the levels of optimization -O2, it is found that the proposed method outperforms optimizations -O2.

Laith H. Alhasnawy 2020[5], To solve the problem of automatic tuning caused by manual tuning compiler optimizations, a machine learning way that relies on the use of a prediction scheme has been proposed. In this paper, K Nearest Neighbor (KNN) classifier algorithm is applied in the prediction scheme to find the optimization passes sequence supported to the features of the program. It also used the reduction algorithm to remove that passes increase time the execution of the program. It was found that the proposed method using the KNN algorithm in the prediction scheme outperforms when compared to the LLVM optimization levels -O2. Zeyd S. Alkaaby 2018[6], In this work, Where was used multi-levels, genetic algorithm has been used to find a good optimal sequence. Our method has three levels. The programs search space is divided into three groups and trying to find the best optimization sequence for each program in the group. Then use these sequences to find a best sequence of all programs in that group. Genetic algorithm will use the resulting sequences to find out one good optimal sequence for all these groups. In general, this approach yields better results when compared with -O2. Suresh Purini 2013 [7], In this approach a completely different technique has been proposed from the iterative compilation and machine learning based prediction techniques. Best optimization sequencing sets have been found, capable of covering many programs in each group. LUIZ G. A. MARTINS 2016 [8], thorough exploration of all optimization sequences is a complex and time consuming task. In this approach, emphasis depends on an efficient Design Space Exploration (DSE) scheme to define the optimization sequence to improve performance of each application function and reduce exploration time. In this approach, a clustering-based selection method is proposed to reduce the number of translator improvements using the DSE approach. It uses a simple and fast algorithm (Clean Algorithm) that significantly reduces DSE execution time compared to the performance improvements achieved by using a Genetic Algorithm (GA). AMIR H. ASHOURI [9], In this article an automatic optimization framework is proposed call MiCOMP, which Mitigates the Compiler Phase-ordering problem. We perform phase ordering of the optimizations using optimization sub-sequences and machine learning. We combined similarities between the problem analyzed and the context of Recommender Systems, and integrated similarity measures to boost exploration efficiency. It was found that this method excels outperforms LLVM's -O3 optimization sequence. Ramani S. 2016 [10] In this work, he proposes a sequence selection algorithm that filters out large optimization sequences on the program area. By doing so, he identifies the best optimization sequences group that reduces the time required to select the best optimization sequence and also reduces the runtime of the program area. The work of this paper uses recommendation system approach to find the better optimization sequences in term of the best passes order.

3 Propose Method

Modern compilers provide many optimization techniques that are used to solve the problem of compiler autotuning. In this paper, a technique inspired from Recommendation System is used. The recommendation is widely used in our daily life, that is a kind of information filtering system, by relying on huge data sets. One of the main approaches of Recommendation System that is called Content Based Filtering (CBF) is used in this paper. The idea of CBF is that the optimizations sequences which performed well on a "similar" (previously explored) program will work well for the unseen program being compiled. The features for running programs are extract from special set of registers equipped by modern processor to measure the performance counter events. Several characteristics of running program are described by these events such as cache hits, cache misses and branch prediction. Table 1 offers the performance counters events where first column lists the perf-events and the second column gives the type of these events.

The training set (50 programs) is used to build a model, and to find the most similar group of programs (NNk) to its unseen one. Since we are dealing with iterative compilation (IC), the best sequences of five programs that the closest to unseen program based on the feature's similarity is chosen as the best

suggestions.
The following is an explication of the proposed method::
1- Extracting the events performance counter for each program used in this method, where 50 events are extract that reflect the program characteristics.
2- Computing the similarity[13] for each program used in this method as follows:

$$Sim(p,pi)=\frac{\sum_{w=1}^{m} pw*piw}{\sqrt{\sum_{w=1}^{m} pw^2}*\sqrt{\sum_{w=1}^{m} piw^2}}\frac{\sum_{w=1}^{m} pw*piw}{\sqrt{\sum_{w=1}^{m} pw^2}*\sqrt{\sum_{w=1}^{m} piw^2}} \quad (1)$$

Table 1 presents the performance counters events

Event	type
Cpu-cycles or cycles, instructions, Cache-references, Cache-misses, Bus_cycles	Hardware event
Cpu_clock(msec), Task_clock(msec), Page-faults OR faults, Context-switches, Cpu-migrations, Alignment-faults, Emulation-faults	Software event
L1-dcache-loads, L1-dcache-loads misses, L1-dcache-stores, L1-dcache-stores misses, L1-icache-loads, L1-icache-loads misses, L1-icache-loads, L1-icache-loads misses, L1-icache- prefetches, L1-icache-prefetches –misses, LLC-load, LLC-loads misses, LLC-strores, LLC-strores misses, LLC-prefetch-misses, Dtlb-loads, Dtlb-loads misses, Dtlb-store, Dtlb-store-misses, Dtlb-prefetches, Dtlb-prefetches -misses, Itlb - loads, Itlb –loads -misses, Branch-loads, Branch-loads-misses ,	Hardware cache event
Sched:sched-stat-runtime , Sched:sched-pi-setprio, Syscalls:sys_enter_socket, Kvmmmu:kvm_mmu_pagetable_walk	Tracepoint event
Stalled_cycles-frontend , Stalled_cycle-backend	Dynamic event
Sched:sched_process_exec, Sched:sched_process_frok, Sched:sched_process_wait, Sched:sched_process_wait_task, Sched:sched_process_exit	Tracepoint event

where p represent the main program and and pi represent the other programs [9].
3- For any new program that is unseen, it is calculated similarity by extracting its features and comparing them with the features of all programs (50 programs)
4- After that, it can be identical to the unseen program with the most similar programs, and since the work is an iterative compilation, the five programs (Top5) that the closest to unseen program is chosen as the best suggestions.
Figure 1 illustrates the suggested method which its main steps listed as follows:
Step 1: Extract Program Features
This way, it used 50 programs where each program that collects 50 events. Next, the similarity of the features of each program with the features of the unseen program is calculated, as in Equation 1.

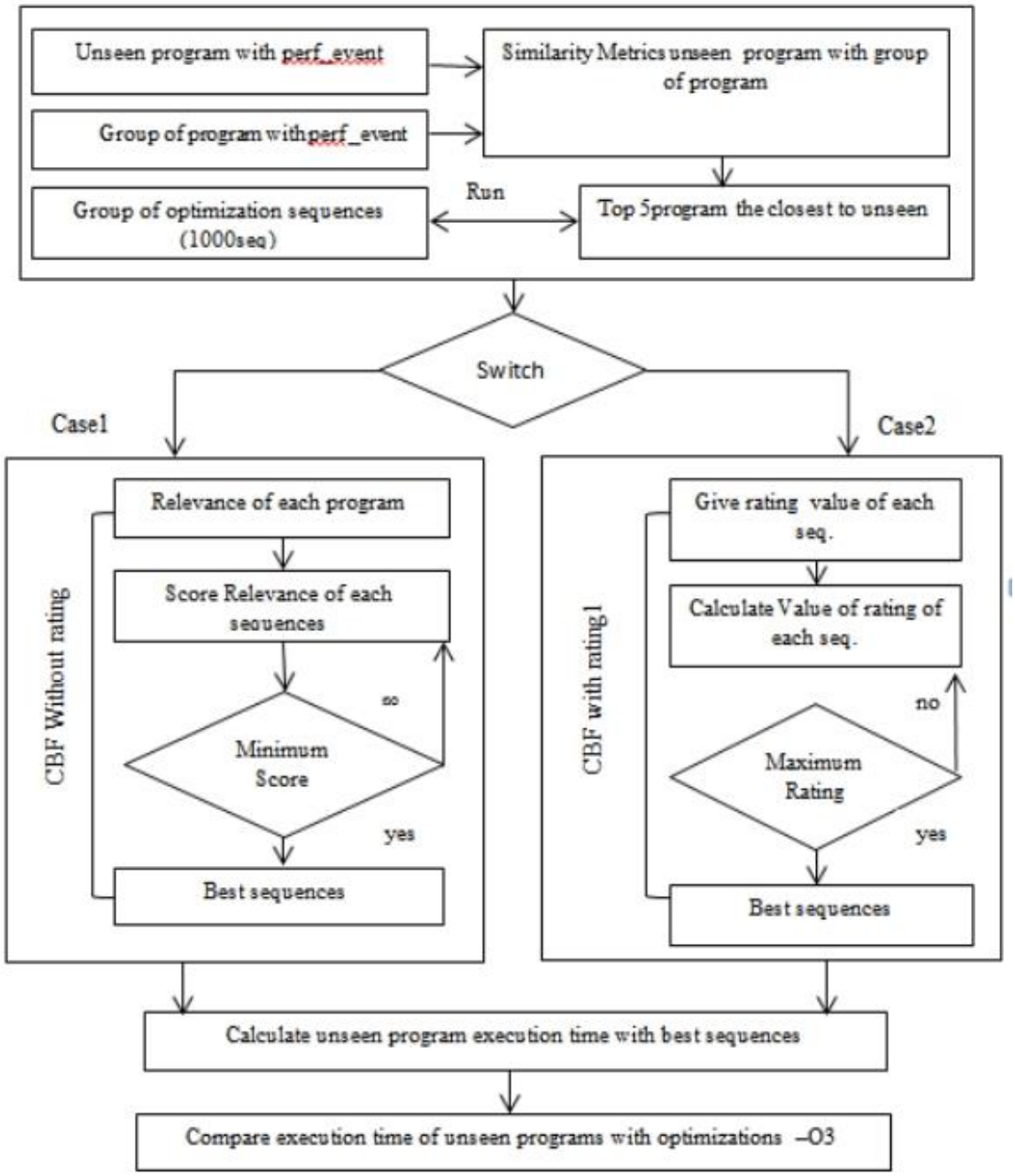

Fig. 1. The structure of the proposed method

The programs are order according to the similarity values; the high value represent the most similar. Depending on the resulting similarity, the programs with the highest similarity ratio are taken (closest 5 neighbors to the unseen program).

Step2: Content Based Filtering without rate value.

Content Based Filtering method is used to find the best optimization sequences that can reduce the execution time for most programs. Thus, compiling the execution time of all the programs with optimization sequence q is collected. Then, the relevance of all programs is calculated according to the Equation 2. Top5 programs that the closest to unseen program are chosen. Thus, each programs p0, p1, p2, . . .,p5 has been compiled and tested with a variety of large set of optimization sequences q0, q1, q2, .

. .q1000(that generated randomly). The relevance rp(q) reflects how much the program p benefits from the optimizations defined in q the baseline set q0 (i.e., -O3), according to the Equation 2

$$rp(q) = \frac{fp(q)}{fp(q0)} - 1 \quad (2)$$

Where *fp(q)* the execution time of the program *p* is compiled using sequence *q*, and *fp(q0)* represents the execution time of the program *p* after it is compiled using sequence of -O3 Flag. The average relevance score for all the programs that compiled with sequence *q* is calculated according to Equation 3. Since, we are dealing with iterative compilation (*IC*) the five sequences that have the less execution time is chosen as the best suggestions.

$$\widetilde{rp}(q) = \widetilde{r}(q) = \left(\frac{\sum p \in P(rp(q))}{|P|} \right) \quad (3)$$

Where *P* represent the all programs.

Step 3: Content Based Filtering with rate value.

In this step, the Content Based Filtering method is working with a rating to choose the best optimization sequences. Where the highest rate values are given to the sequences that have the best execution time, and the best sequence with the highest rate value is suggested. More specifics about the proposed method are offered as follows:

1- Distant values are calculated (D ρ (q_i)), which is the amount of difference between program (P) execution time optimized using sequence (qi) and the same program (P) optimal with the -O3 flag. According to the following equation: (Equation 4)

$$Dp(qi) = execution\_timeP(qi) - execution\_timeP(- \quad (4)$$

2- A rate value is suggested for each sequence depending on the value of Distant *(D ρ (q_i))*. The lowest Distant value *(D ρ (q_i))* is given the highest rating.

3- For each optimization sequence, we collect the rate values that given by all the programs to that sequence. We choose the sequence with the highest rate value.

As we are dealing with an Iterative compilation, five sequences with, highest rate values are selected. Giving each of the five sequences a value of rate between 1 to 5. Thus, the rate value =5 is given to the best sequence (less execution time), rate vale=4 for the next sequence, and so on.

Step 4: The evaluation step.

In this step, evaluation is done on unseen programs. Where the unseen programs are executed with the best five sequences, the ones giving lowest execution time, that generate from the step2 (without rating). Then, this process is repeated again where the unseen programs are executed with the best five sequences generate from the step 3 (with rate value). The results of the first step and second step are compared with the optimization -O3 results. The algorithm below illustrates the proposed method.

Algorithm (1): Content Based Filtering algorithm for best optimization sequence

Input : 50 different programs, vector of 50 event

Output: best five recommended optimization sequences
1*: For* i=1 to number of programs
Features extraction from program
End For I
2*:* ***For*** *j* = 1 to number of programs
Find to classify program[j] for the closest unseen programs by its features
End For j
3: *For* i=0 to number of optimization sequences
Seq[i] à Generate random optimization sequences by selecting passes from passes vector.
4: *For* *i*=0 to number of optimization sequences
For *j*=0 to number of programs that closest to unseen
Exe_time[i]. p[j] à Calculate the time of execution of programs closest to the unseen program
5: *Switch* (variable): {
// best optimization sequences without rating
Case(1)*:*
1: ***For*** i=0 to number of optimization sequences
For j=0 to number of programs that closest to unseen
Exe_time[i]. relevance[j]=(exe_time[i]. p[j]/exe_time. p_O3[j])-1
2: *For* i=0 to number of optimization sequences
Sum=0
For j=0 to number of programs
Sum=sum+ Exe_time[i]. relevance[j]
Exe_time[i].score_relevance = sum /number of Programs
3: Select minimum (Exe_time[i]. score_relevance) à best optimization sequences
// best optimization sequences with rating.
Case(2):
1:***For*** i=0 to number of optimization sequences
For j=0 to number of programs that closest to unseen
Exe_time[i].Dist[j]= exe_time[i].p[j]-exe_time.p_O3[j]
2:*For* i=0 to number of programs that closest to unseen
For j=0 to number of optimization sequences
minimum (Exe_time[i]. Dist[j]) à rating
//Give the rating of each programs according to their optimization (Dist).

3: For each optimization sequences, rating values are calculated for all programs in that sequences
4: Maximum rating values à best optimization sequences and give rate.
Default: *exit;*
}
5:Test of unseen programs with best optimization sequences (case1 ,case2) and calculate execution time.
Compare execution time(**case1,case2**) with optimization sequences -O3.

4 Experimental Evaluation

This part discusses the results that obtained from the implementation of the proposed method. The description of how the dataset is built is introduced in section 4.1. In section 4.2, the proposed method is evaluated. Table 4 summarizes the technical details of the evaluation platform.

TABLE 1. Platform details

Processor type	Intel core-i7
Processor speed	1.80GHz
Processor	1 CPU, 4 Core, 2 threads per Core
L1d Cache size	32k
L1i Cache size	32k
L3 Cache size	8192k
RAM	4 GB

Operating System	Ubuntu 16.

TABLE 1. Platform details

4. 1. Dataset collection

Our experiments build a dataset of 50 different programs chosen from three different benchmark suite named PolyBench, Shootout and Stanford. The work is done to find a performance counter events for each program using perf Linux tool. And then finding the programs closest to the unseen program. These programs are compiled with 1000 randomly generated optimization sequences. Each sequence has generated different lengths from 1 to 60 passes. Each of the program is three times executed to obtain the most accurate results for capturing the average execution time. More details about building a database as follows:

1- Extract program features using performance counter events.

Performance counter events can describe the characteristic of running program. Where 50 events are extract that reflect the dynamic behaviors of these programs [12].

2- Generating a random optimizations sequences of different lengths.

The best three sequences that are obtained from [4] are included in our data set. The rest of the sequences are randomly generated from 60 LLVM passes showed in Table 2. Finally, the total number of the generated sequences is equal to 1000.

TABLE 2. Optimization passes of -O3 standard optimization level

List of -O3 optimization passes		
-domtree	-lcssa-verification	-early-cse-memssa
-inline	-loop-accesses	-rpo-functionattrs
-scalarizer	-globaldce	-prune-eh
-called-value-propagaton	-loop-load-elim	-ipsccp
-tti	-inferattrs	-globaldce
-assumption-cache-tracker	-div-rem-pairs	-indvars
-opt-remark-emitter	-profile-summary	-forceattrs
-lazy-block-freq	-tbaa	-mem2reg
-block-freq	-libcalls-shrinkwrap	-globalopt
-instsimplify	-jump-threading	-tailcallelim
-loop-unswitch	-globals-aa	-instcombine
-licm	-targetlibinfo	-reassociate
-simplifycfg	-scalar-evolution	
-memoryssa	-basiccg	
-loop-rotate	-loop-simplify	
-callsite-splitting	-speculative-execution	
-aa	-loops	
-demended-bits	-sroa	
-loop-unroll	-basicaa	
-pgo-memop-opt	-lazy-block-free	

3-The execution time of all programs.

The group of programs are evaluated where each program (*p*) is compiled with all optimization sequences created in step (1). The execution time of program *p* that is compiled using sequence *q* is denoted by *fp(q)* refer with (Equation 1).

4- O3 program execution time.

All the programs are compiled with -O3 flag which represents the stander optimization sequence and it is denoted by *fp(q0)* refer with (Equation 1).

4. 2. Experimental Results

The results of applying the *CBF* method are presented in this section where 50 programs are used to build the dataset and 5 unseen programs are used to evaluate the proposed method. A set of best optimization sequences was recommended as final obtained results. The comparison is made between the set of best optimization sequences (CBF without rate value) and -O3 flag. Figure 1 shows the comparisons between the execution time for set of unseen programs compiled with -O3 flag and the same set compiled with best

five sequences (wrec_seq1, wrec_seq2, wrec_seq3, wrec_seq4, wrec_seq5) resulted from Content Based Filtering method without rate value. Moreover, it shows that for most of these programs the CBF method outperforms the -O3 optimization sequence of a factor 9.3% in term of reduction in the execution time.

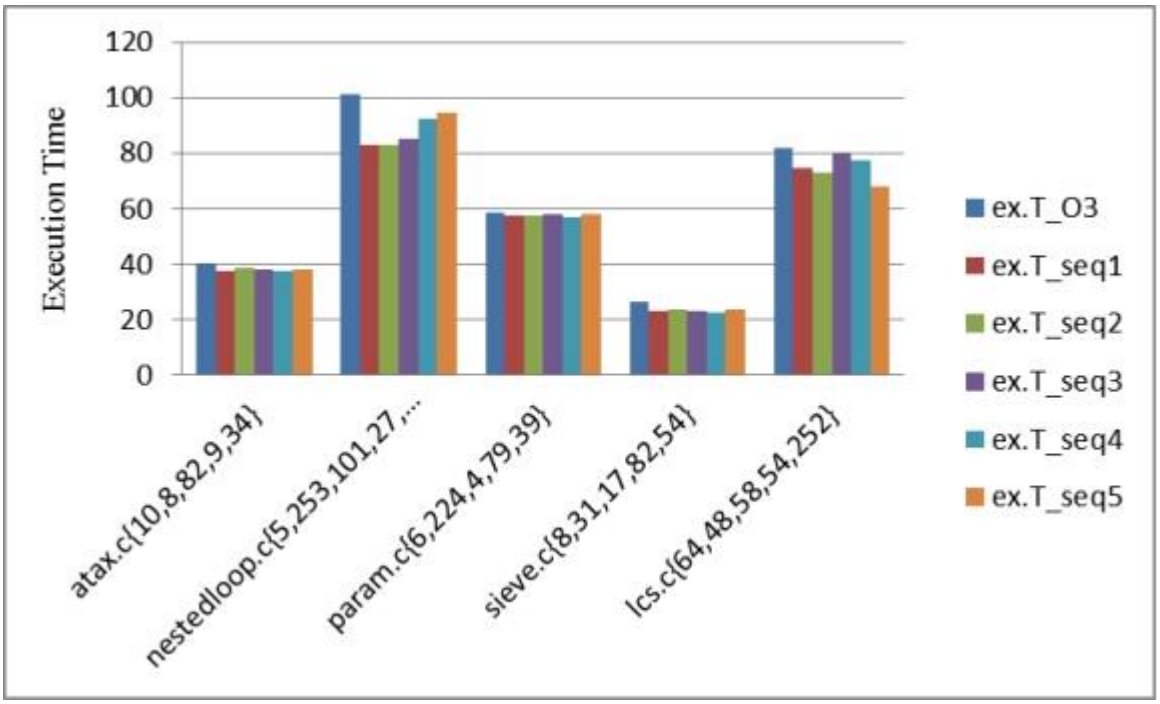

Fig. 1. Execution time of unseen programs (CBF without rate value)

In Fig. 1. the X AXIS represents the unseen programs (5 unseen programs) that was tested using the method CBF without rating and obtained the best 5 different optimization sequences for each unseen program (The numbers in the parentheses of the group represent the numbers of the top 5 improvement sequences obtained out of 1000 improvement strings (which we randomly generated)) and compare with -O3 flag . As for the Y AXIS, it represents the time of execution of each unseen program with the -O3 flag and (5TOP) the best different optimization sequences of obtained.

4. 2. 1. The results of the CBF with rate value.

In this approach, the results of applying the CBF method with rate value are presented. The sets of best recommended optimization sequences were identified for comparison. The results of compilation of the best five sequences over a set of unseen programs are compared to the results of the same set of unseen programs compiled with -O3 optimization sequence.

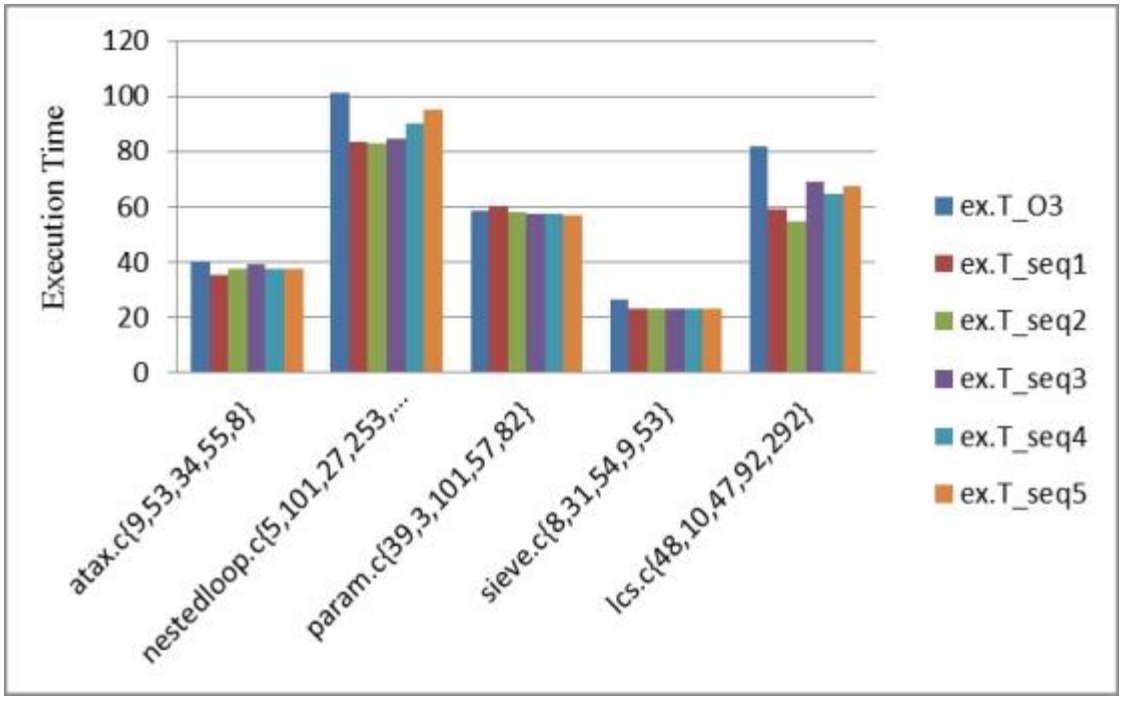

Fig. 2. Execution time of unseen programs (CBF with rate value)

In Fig. 2. the X AXIS represents the unseen programs (5 unseen programs) that was tested using the method CBF with rating and obtained the best 5 different optimization sequences for each unseen program (The numbers in the parentheses of the group represent the numbers of the top 5 improvement sequences obtained out of 1000 improvement strings (which we randomly generated)) and compare with -O3 flag . As for the y axis, it represents the time of execution of each unseen program with the -O3 flag and (5TOP) the best different optimization sequences of obtained. The results show that on average for most of these programs outperforms the -O3 optimization sequence of a factor 13.7% in term of reduction in the execution time.

5 Conclusion

This paper presented a method that provides autotuning compiler optimization instead of manual tuning. Content Based filtering (CBF) method is used to find the best optimization sequences that can reduce the execution time for unseen program based on the similarity measure. Two versions of CBF method (with and without rate value) were proposed. Thus, 50 different programs and 1000 randomly generated sequences are used to build the dataset. The results of two versions that were evaluated using 5 unseen programs showed the performance of the proposed method in improving the time of execution by average of 9.3% for CBF without rate, 13,7% for CBF with rate in term of execution time reduction.

6 References

1. Amir Hossein Ashouri, Giovanni Mariani, Gianluca Palermo, Eunjung Park, John Cavazos and Cristina Silvano: COBAYN: Compiler Autotuning using Bayesian Networks ACM Trans. Architec. Code Optim. 0, N, Article A (2015), 25 pages.
2. Leif Uhsadel, Andy Georges, Ingrid Verbauwhedey: Exploiting Hardware Performance Counters. p.59-67(2008).
3. Enneth Hoste and Lieven Eeckhout : Cole: compiler optimization level exploration. In Proceedings of the 6th annual IEEE/ACM international symposium on Code generation and optimization. ACM, 165–174. 2008, Boston, Massachusetts, USA.
4. Manal H. Almohammed, Ahmed B. M. Fanfakh(B), and Esraa H. Alwan. Parallel Genetic Algorithm for Optimizing Compiler Sequences Ordering. NTICT 2020, CCIS 1183, pp. 128–138, 2020.
5. Laith H. Alhasnawy, Esraa H. Alwan, and Ahmed B. M. Fanfakh. Using Machine Learning to Predictthe Sequences of Optimization Passes. NTICT 2020, CCIS 1183, pp. 139–156, 2020.
6. Zaid S. Alkaaby, Esraa H. Alwan, and Ahmed B. M. Fanfakh: Finding a Good Global Sequence using Multi-Level Genetic Algorithm. Journal of Engineering and applied Science 13 (22):9777-9783, 2018.

7. Purini, S., Jain, L.: Finding good optimization sequences covering program space. ACM Trans. Archit. Code Opt. (TACO) 9(4), 56 (2013).
8. Luiz G. A. Martins, Ricardo Nobre, Jo˜ao M. P. Cardoso, Alexandre C. B. Delbem, and Eduardo Marques: Clustering-Based Selection for the Exploration of Compiler Optimization Sequences. ACM Trans. Archit. Code Optim. 13, 1, Article 8 (March 2016), 28 pages.
9. Ashouri, A.H.,Bignoli, A., Palermo, G., Silvano, C.,Kulkarni, S.,Cavazos, J.: Mitigating the compiler phase-ordering problem using optimization sub-sequences and machine learning (MiCOMP). ACM Trans. Archit. Code Opt. (TACO) 14(3), 29 (2017).
10. Ramani S, Boominathan P, Swathi J, Narayanana , Rajkumar S.: Improving Optimization Sequence of Compilers by Using Sequence Selection Approach. IJPT| June-2016 | Vol. 8 | Issue No.2 | 13471-13480.
11. John Cavazos1 Grigori Fursin2 Felix Agakov1 Edwin Bonilla1: Rapidly Selecting Good Compiler Optimizations using Performance Counters.
12. Nisbet, A.P.: GAPS: Iterative feedback directed parallelization using genetic algorithms. In: Workshop on Profile and Feedback-Directed Compilation (1998).
13. Kumar, T.S., Sakthivel, S., Kumar, S.: Optimizing code by selecting compiler flags using parallel genetic algorithm on multicore CPUs. Int. J. Eng. Technol. (IJET) 6, 544–555 (2014).
14. Stefano Cereda, Gianluca Palermo, Paolo Cremonesi and Stefano Doni: A Collaborative Filtering Approach for the Automatic Tuning of Compiler Optimisations. In Proceedings of 21st ACMSIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems, London, United Kingdom, June 16, 2020 (LCTES ’20), 10 pages.