Improving Code Efficiency by Code Optimising Techniques

Neeraj Kumar [1], Prof. Dr. Saroj Hiranwal [2]

[1]*M.Tech Scholar, Dept. of Computer Science & Engineering, Rajasthan Institute of Engineering and Technology, Jaipur, Rajasthan, INDIA*

[2]*Professor, Dept. of Computer Science & Engineering, Rajasthan Institute of Engineering and Technology, Jaipur, Rajasthan,*

---***---

Abstract - *In this paper we presented a tool which optimizes the code by working on the following segments of code optimization namely Dead Code Removal, Inlining, and Constant Propagation etc. The change in the nature of code remains a major issue from the prior days. Once in a while it is troublesome for a software engineer to discover which some portion of code devours more assets and thus prompt a wasteful code. Already the vast majority of the enhancement was done physically or can be said as statically which prompts number of issues to the developer furthermore had a portion of the constraints. Be that as it may, nowadays a few compilers are accessible which makes the streamlining to be performed progressively. In this paper work, an endeavor has been made to plan and execute a framework that can consequently improve the code keeping in mind the end goal to minimize the multifaceted nature of the code such that the code turns out to be more proficient. To finish this, a framework has been created to execute the two machine autonomous methods dead code end and normal sub-expression disposal, which powerfully improves the code. The postulation essentially moves around these two procedures, how these methods have been executed, and how can it works alongside the figuring of the code intricacy. For this, the codes were taken haphazardly. Subsequently the advancement of code could be conceivable utilizing diverse enhancement methods. Rather than concentrating on adding to another calculation or to enhance the current one, this paper endeavors to comprehend the current compiler strategies unmistakably and to examine the outcome subsequent to actualizing the systems furthermore to think about the complexities taking into account diverse measurements of the code.*

Key Words: *Code Optimization, Code Improvement, Code Enhancement, CCCC, LOC, McCabe's Cyclomatic number*

1. INTRODUCTION

In compiler outline, there is one of the strategy in which a piece of code is being changed to create more proficient and additionally to enhance the execution such that the yield stays same, termed as "Improvement". Code improvement intends to make superb code with best intricacy (time and space) such that it ought not influence the precise consequence of the code. It is essentially taking into account the basis to safeguard the semantic proportionality of the project, such that the calculation must not be adjusted. On a normal, the change ought to accelerate the execution of the system. Improvement incorporates finding a bottleneck, a basic part of the code which is the essential customer of the required assets. Essentially Code streamlining worries on rightness, it implies the accuracy of the created code ought not to be changed. The primary point of the code enhancement is to make fantastic code with enhanced many-sided quality (time and space) without influencing the accurate consequence of the code.

2. IMPORTANCE AND RELEVANCE OF THE STUDY

Enhancement is that the field wherever the majority of the examination is finished. Looking the different papers and article some of the significant information are assembled that makes the advancement strategy doable. [5]

Chirag [3] depicts concerning the gap improvement method utilizing totally diverse example coordinating methodologies that structures A standard expression and conjointly has investigates the past and current examination issues in term of "advancing" compilers utilizing enhancement decides that range unit envisioned to be coordinated through that the excess guideline of moderate code are frequently researched and supplanted. For this totally distinctive example coordinating methodologies are said like string based generally, tree control, object based for the most part and so forth. The paper is being gap into four segments wherever particular guidelines and example standards are difficult coded that clarifies concerning the machine subordinate gap streamlining agents, the motivation of retarget prepared gap enhancement the mix of code era and advancement into one section and concerning totally distinctive example coordinating strategies severally.

Brandolese[7] presented a thorough system for programming bundle execution time estimation, that is upheld by thorough numerical models of C articulations as far as basic operations. Amid this the complete investigation stream has been performed inside an encapsulation toolset that are in the blink of an eye being utilized for steady and adjusting model parameters.

Johnson[6] in his article portrays concerning the advancement that is system of changing a touch of code such the code gets to be extra practical and in this way the yield stays same. it's as a rule conjointly announced that a considerable measure of the issues were NP finished and in this manner a large portion of the enhancement recipe relies on upon guess and heuristics. Conjointly he announced that when the code has been composed, given the compiler a chance to do the enhancement utilizing compiler procedures.

In this paper the layout is given concerning the improvement that concentrates consequently the accumulation technique should be similar to on the grounds that the rightness of the produced code mustn't be altered, conjointly procedure is characterized like "when" and "where" to advance for playing enhancement. some of the procedures connected to middle of the road code, others zone unit connected to conclusive code era and even some of the strategies might happen once the last code era inside of which the attempt is made to redesign the gather code itself to extra sparing one. It conjointly depicts concerning the "neighborhood advancement" that is delineated in light of the fact that the enhancement that zone unit exclusively authorized inside a fundamental piece, in this manner these kind of improvement region unit simple to execute as a consequence of any sensibly administration information isn't required exclusively the streamlining is to be performed inside a square. Some of the local advancement such as steady collapsing, consistent proliferation additionally are clarified. Advancements that might take out futile guidelines utilizing logarithmically personalities range unit like administrator quality decrease, duplicate spread, and dead code disposal. With the exception of local improvements, the same sensibly streamlining that may be connected over the fundamental squares makes them world advancement. Conjointly concerning machine streamlining is clarified; one among the machine advancement that is of particular significance is register allotment, another most noteworthy enhancement is direction programming.

Michael E Lee [4] narrates methods which were utilized to advance the code of C. The fixation is on lessening time spent by the CPU and gives test source code change which frequently yield change. The change or advancement is performed such that the engineers of the application programs have the obligation to plan programs with a specific end goal to make utilization of restricted and costly assets.

3. Proposed Solution

System architecture describes about how the system works and how the process flows. The layout of the work is presented that shows how the transformation takes place. Initially, starting with the selection of desired technique which is to be performed, then it is required to select the program code which needs to be optimized or the code to perform transformation. After selecting the program code (unoptimized code), now the need is to apply the technique which was selected. When the button, provided for the same purpose is clicked then the attribute of program structure matches with the techniques which is applied i.e. if the selected technique will work on the chosen program code then the new optimized code will be generated, and if not then an warning message will be displayed, or the errors message which were encountered during the implementation of the technique and the control will transfer again to select the new code. If the optimization will take place then the new optimized code needs to be saved. Once the optimized code is saved, complexity comparison is performed between the program source codes with the new generated optimized code.

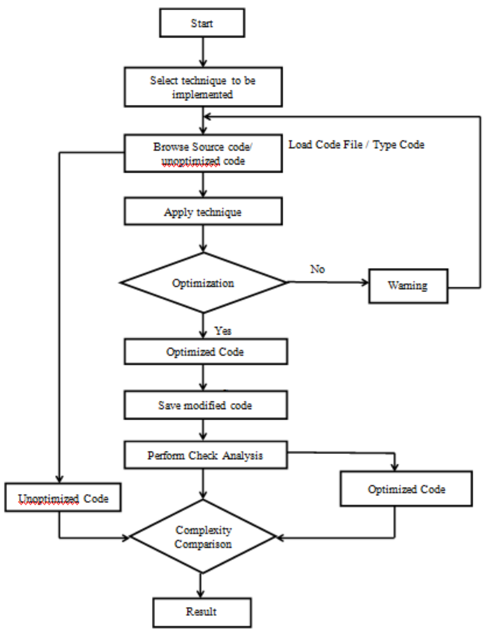


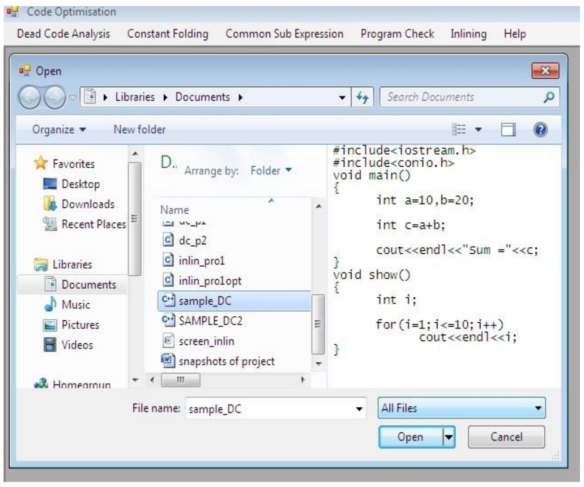
Fig – 1: Proposed System Architecture

4. Result & Findings

This section of the thesis describes about the results or findings which is obtained after the implementation of optimization techniques. Two elementary objectives are to implement the optimization techniques such that sample code should optimize automatically i.e. the dynamic optimization of code takes place and the resultant code will be one of the new optimized code and to compare and analyze the complexity of unoptimized code with the new one generated optimized code in order to reduce the complexity of the code. The objectives where achieved. The analysis and comparison between the program code and resultant code are done using CCCC tool in a report which is shown below.

4.1 Result showing transformation of unoptimized code into optimized one and complexity comparison between the program code and target code using Dead Code Elimination:

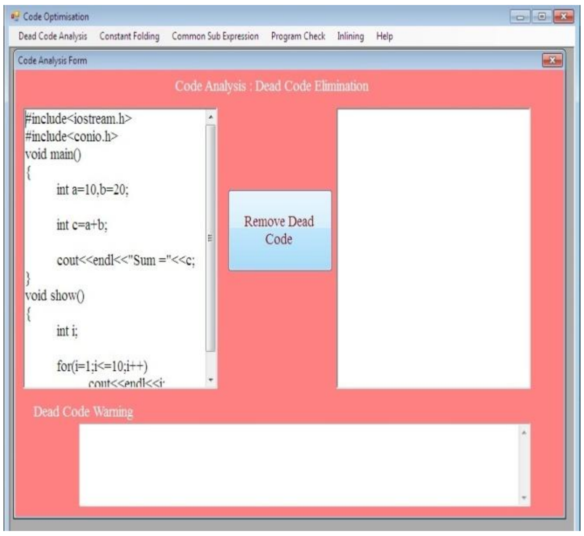
This part of result shows the transformation which takes place after applying the dead code elimination. Different snapshots, windows were presented which shows how the code is been initially selected using browse option to be further optimize, and how the elimination is performed, and also the window displaying report of different metrics of both the unoptimized and optimized code.



```
#include<iostream.h>
#include<conio.h>
void main()
{
        int a=10,b=20;

        int c=a+b;

        cout<<endl<<"Sum ="<<c;
}
void show()
{
        int i;

        for(i=1;i<=10;i++)
              cout<<endl<<i;
}
```

Fig - 2: Selection of the program code

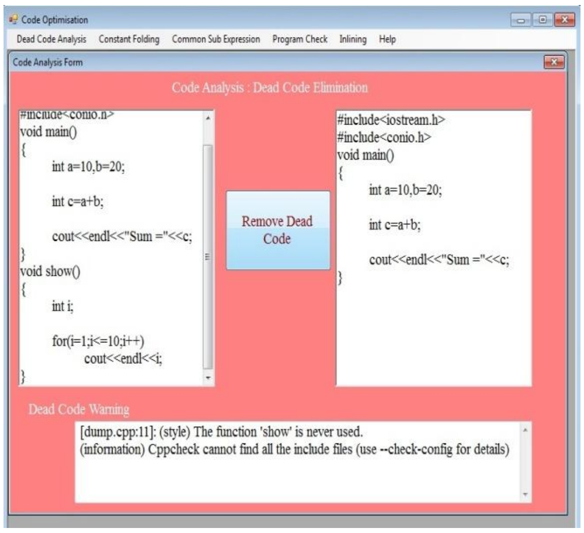
The above given snapshot in figure 2 shows how to browse for the code file in order to perform optimization using respective technique. The selected code file is used for the dead code elimination.



```
#include<iostream.h>
#include<conio.h>
void main()
{
        int a=10,b=20;

        int c=a+b;

        cout<<endl<<"Sum ="<<c;
}
void show()
{
        int i;

        for(i=1;i<=10;i++)
              cout<<endl<<i;
```

Fig - 3: Importing program code for optimization

When the desired file is browsed, the source code opens up into a text box in the window for the dead code analysis. This is well understood by the snapshot in the figure 3.



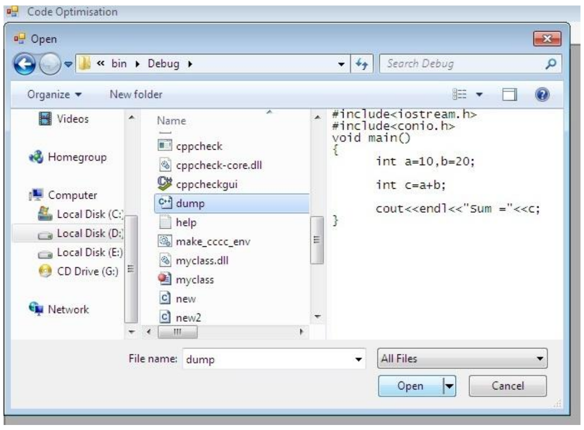
```
#include<conio.h>
void main()
{
        int a=10,b=20;

        int c=a+b;

        cout<<endl<<"Sum ="<<c;
}
void show()
{
        int i;

        for(i=1;i<=10;i++)
              cout<<endl<<i;
}
```

```
#include<iostream.h>
#include<conio.h>
void main()
{
        int a=10,b=20;

        int c=a+b;

        cout<<endl<<"Sum ="<<c;
}
```

```
[dump.cpp:11]: (style) The function 'show' is never used.
(information) Cppcheck cannot find all the include files (use --check-config for details)
```

Fig – 4: Window showing the removal of unused function

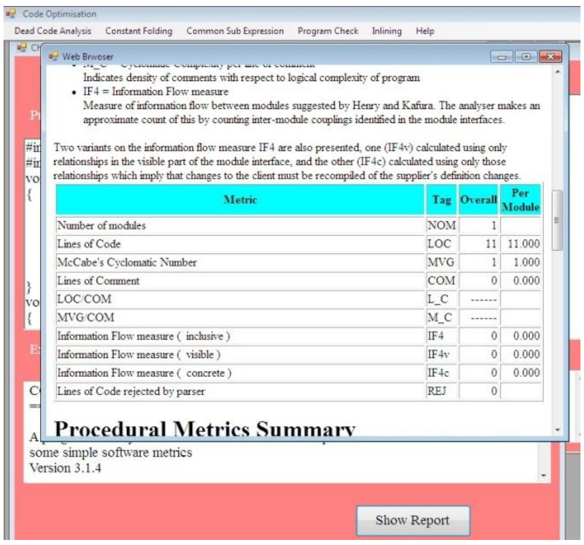
Figure 4 shows the transformation which takes place after clicking on button named remove dead code. The function which was not used anywhere in the program got eliminated and dead code elimination is performed. The optimized i.e. the target code is shown in the right hand side of rich text box which automatically saves in debug folder with the name dump.cpp. The below given snapshot shows the file with name dump.cpp where optimized code is saved.



```
#include<iostream.h>
#include<conio.h>
void main()
{
        int a=10,b=20;
        int c=a+b;
        cout<<endl<<"Sum ="<<c;
}
```

Fig – 5: Result saving window for resultant code

After the implementation of the dead code elimination, the resultant code is saved in the bin folder, which could be seen as in figure 5.



Indicates density of comments with respect to logical complexity of program

- IF4 = Information Flow measure
  Measure of information flow between modules suggested by Henry and Kafura. The analyser makes an approximate count of this by counting inter-module couplings identified in the module interfaces.

Two variants on the information flow measure IF4 are also presented, one (IF4v) calculated using only relationships in the visible part of the module interface, and the other (IF4c) calculated using only those relationships which imply that changes to the client must be recompiled of the supplier's definition changes.

Metric	Tag	Overall	Per Module
Number of modules	NOM	1	
Lines of Code	LOC	11	11.000
McCabe's Cyclomatic Number	MVG	1	1.000
Lines of Comment	COM	0	0.000
LOC/COM	L_C	------	
MVG/COM	M_C	------	
Information Flow measure (inclusive)	IF4	0	0.000
Information Flow measure (visible)	IF4v	0	0.000
Information Flow measure (concrete)	IF4c	0	0.000
Lines of Code rejected by parser	REJ	0	

Procedural Metrics Summary

some simple software metrics
Version 3.1.4

Fig – 6: Report generated by the CCCC of unoptimized code

This window shows the report which is generated by the CCCC tool when the program is analyzed for the complexity. Different metrics were displayed showing number of modules, LOC and McCabe's Cyclomatic Number. Here, the analysis of source code has been done where the lines of code are 11 and McCabe's Cyclomatic number is 1.

4. Conclusion and Future Scope

The paper work focuses to apply the optimization techniques on the C/C++ codes in-order to improve the complexity of the source code with respect to size and time. The machine independent techniques have been implemented successfully as desired with some limitations on the source codes used. The interface designed for this work consists of the various modules through which the work has been fulfilled. Different techniques have their own way of modifying the source code and attempt to optimize the code which reduces the complexity of that code. Dead code analysis finds the functions that are not required in the whole program and so those functions are eliminated. Common sub-expression elimination replaces the multiple instances of the common expressions with a new variable, so that the computation is performed only once and its value could be replaced with all other instances of the common expression. The complete analysis has been done on the interface specially designed for this work. The implementation process has been shown as how the technique has been applied on the code and the new modified code is produced. Then the complexity has been calculated first for the original source code and then for the new modified code. This complexity is calculated using CCCC tool based on various metrics. When the comparison of the complexity of the original code and the modified code is performed, it results into the reduction of the complexity. The goal that was proposed earlier has been accomplished completely.

The future scope still exists for this work which includes few modifications that would make it more functional. Firstly, the source codes on which the techniques are applied have some limitations depending on the technique used. Like dead code analysis module only attempts to remove those functions that are not used in the main calling function. Moreover, the common sub-expression elimination module only attempts to replace a single common expression only. These are some limitations of the modules that need to overcome. Secondly, this work could be more effective if a compiler could be incorporated within the interface so that the output of the source codes must also be displayed. Finally, the advancements would be beneficial such as the comparison of the complexities of both the un-optimized code and the optimized code could be displayed on the same screen.

REFERENCES

[1] Mr. Chirag H. Bhatt, Dr. Harshad B. Bhadka , "Peephole Optimization Technique for analysis and review of Compile Design and Construction", IOSR Journal of Computer Engineering (IOSR-JCE), Volume 9, Issue 4 (Mar. - Apr. 2013).

[2] C. Brandolese, W. Fornaciari, F. Salice, D. Sciuto, "Source–Level Execution Time Estimation of C Programs", Proceedings of the ninth international symposium on Hardware/software codesign.

[3] Maggie Johnson, "Code Optimization", Handout 20 "August 08,2004.

[4] Michael E. Lee, "Optimization of Computer Programs in C ", Ontek Corporation, USA.

[5] Keith D. Cooper, L. Taylor Simpson, Christopher A. Vick" "Operator Strength reduction" available at :http://www.cs.rice.edu/~keith/EMBED/OSR.pdf.

[6] Tom Erkinen,"Fixed point ECU code optimization and verification with model based design" available at http://in.mathworks.com/tagteam/59064_2009-01-0269.New.pdf.

[7] Paul Heish, "Programming Optimization: Techniques, examples and discussion" available at: http://www.azillionmonkeys.com/qed/optimize.html

[8] Keith D. Cooper, Kathryn S. Mckinley, and Linda Torczon,"Compiler-Based Code-Improvement Techniques" available at http://www.cs.tufts.edu/~nr/cs257/archive/keith-cooper/survey.pdf

[9] Req. Charney, "Programming Tools: Code Complexity Metrics" available at http://www.linuxjournal.com/article/8035

[10] Bruce Childers, Jack W. Davidson, Mary Lou Soffa, "Continuous Compilation: A New Approach to Aggressive and Adaptive Code Transformation" available at http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=1213375&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D1213375\

[11] Doeppner, "Optimization Techniques in C", Fall, 2013. http://cs.brown.edu/courses/cs033/docs/guides/c_optimization_notes.pdf

[12] Mohammed Fadle Abdulla, "Manual and Fast C Code Optimization", Anale. SeriaInformatica. Vol.VIII fasc. I-2010.

BIOGRAPHIES

Neeraj Kumar is a Research Scholar from Rajasthan Institute of Engineering and Technology. He has BE degree from the University of Rajasthan, Jaipur. His area of interest including Software Engineering, Computer Graphics, Data Structures and Computer Networks.

Prof. Dr. Saroj Hiranwal is a Professor in the Department of Computer Science and Engineering in Rajasthan Institute of Engineering and Technology, Jaipur. She has completed her Doctorate degree from Suresh Gyan Vihar University, Jaipur. Her area of interest includes Computer Graphics, Distributed System, Software Engineering and Computer Architecture.