International Journal of Research in Engineering, Science and Management
Volume-2, Issue-4, April-2019
www.ijresm.com | ISSN (Online): 2581-5792

Hybrid Code Optimization Technology

M. Supriya[1], B. K. Harini[2], Sini Anna Alex[3]

[1,2]*Student, Department of Computer Science and Engg., M. S. Ramaiah Institute of Technology, Bangalore, India*

[3]*Assistant Professor, Dept. of Computer Science and Engg., M. S. Ramaiah Inst. of Tech., Bangalore, India*

***Abstract*: Real time application must be very fast and reliable because it as to respond as quickly as possible. To do so the code which as to execute to perform that operation or task should be clear and optimized. But while coding programmer can't concentrate much on optimized code so this task as to assign for compiler so that while compiling the code it goes through the code once to analyze is there any code which can be optimized by applying optimization techniques. In this paper we moved one step forward and we are grouping these optimization techniques so that instead of applying only one or two optimization techniques apply all and output a good optimized code to run more fast and efficiently.**

***Keywords*: Optimization Techniques, Applications, Coding.**

1. Introduction

Now-a-days in the competitive world of technology code has been used in many of the real time application. Code plays a vital role in any of the development works. But due to some of the reasons that the code being very large, time consuming, debugging, resource consumption requires to be optimized. Optimization is the technique for transforming the code which intern improves delivery speed and resource consumption. Optimization usually represent program in the lower level for better understandability and convenience purpose. In this we are focusing on combining the code optimization techniques which overcomes the existing disadvantages of the normal code. Optimization must and should result in correct semantics of the program, minimum resource consumption and faster in optimizing the code itself.

2. Related works and discussions

A. Literature Survey

This paper mainly focuses on optimizing loops. Loops consumes more resource and time to get execute so to make them consume less resource and less time to execute the LLVM (Low Level Virtual Machine) namely loop fusion. Loop fusion combines the two loops by adding the second loop to the first loop by deleting the second loop. But to do this some conditions has to fallow i.e. two loops must contain same number of iterations, there should not be any code between two loops, data between two loops must be independent.

By doing this we can reduce the number of thread creations to execute loops iteratively so by decreasing the number of threads energy consumption can be reduced and program can be execute fast.

LLVM is designed as a modular system to compile C/C++ programming language. In this system can enable optimizations level of compiling by partially vectorized loops i.e the loops can be execute in parallel manner if a system is a multiprocessor system. So that the it improves the temporal and spatial locality. Loop fusion reduces the cache miss rate and increases the hit time and hit rate. Finally, LLVM optimizes the loops by using mathematical compile time transformations steps called passes these passes are built by PassBuilder class. Loop fusion algorithm decreases the number of loops by merging them and by applying parallelization and vectorization increases the performance on multi-core architectures. [1]

Another survey is made by Paul B Schneck on compiler optimization techniques. It describes the optimization techniques and their grouping. They are grouped into three categories as machine dependent, architecture dependent and architecture independent.

Machine dependent architecture are basically used to reduce the time and the program space and they are used and implemented locally and are performed on short time generated codes with some applicable properties. In these optimization techniques the instructions have been considered. In the architecture dependent optimization, the instructions are not been considered and are used globally. The implementation takes place while generating the code or the program. The structure of the compiler is the main consideration of the architecture dependent optimization.

The architecture independent optimization is similar to the architecture dependent optimization but they rely on the dependencies and the flow between the codes in the program. This paper also includes the review about the universal optimization. They finally conclude that the architecture independent optimization is responsible for the increased speed of the compilers [3].

B. Overview of the existing works

Some of the code optimization techniques are as follows:

1. Basic blocks
2. Common sub expression elimination
3. Control flow graph
4. Dead code elimination
5. Loop optimization

1) Basic blocks

Codes are generally executed in the linear sequence i.e. step by step execution. They are said to be the basic blocks of the code. The compiler decomposes the code into the number of blocks. Normal codes have the jump statements which are said to have more computation but these blocks doesn't involve in any of the go to or jump statements.

When the execution starts from the first line a block will be created for the line and flow will be maintained till the completion of the respective block. Once a block is created for the particular code there would be no permission for the other codes to enter the block or exit the block.

The code inside the block will be executed exactly once. But what about the jump or loop statements in blocks? The loops or jump statement will be created as the separate block so that there wouldn't be any interruption for the normal codes by these statements. This helps in faster execution of the codes.

2) Common sub expression elimination

The expressions which already been computed will be appeared in the code again and again then those expressions can be called as common sub expression. As the name itself suggest that a sub expression which is commonly used will be identified. After identification of the expression the value will be computed. The value assigned to the identified expression will be replaced wherever being used in the entire program. The common sub expressing can be classified into local and global common sub expressions. Local sub expression where the search for the expression will be restricted for the particular blocks where as in global common sub expression searches for the expression in the entire program or the procedure. This method would help in reducing the computation and lesser memory usage [6].

3) Control flow graph

Here in this the blocks will be represented as nodes. Control flow graphs gives the information about where to go next or the flow to be followed further i.e. control information. This control information will be given inside the blocks itself to avoid confusion. It helps in providing the connectivity between the nodes i.e. blocks.

4) Dead code elimination

There are some codes which is never used or not reachable or in case used their output will not be useful. Those codes can be called as dead codes. So, by eliminating such codes will be highly advantageous. It would decrease the code size, lesser execution time, minimized memory usage [7].

5) Loop optimization

Loops usually eats up a larger time for execution and more computations. There are some codes in the loops which produces the same output even inside or outside the loop. So, it is better to maintain such codes outside the loops. Sometimes programs tend to execute more loops. Better to avoid loops as much as possible to improve accuracy. Loop optimization is the process of increasing the execution speed and reduce the overheads introduced by the loops [5].

C. Proposed system

In the proposed system the program will be divided into number of blocks. The blocks will be represented in the form of nodes. The flow between the nodes i.e. control information will be provided by the control flow directed graph. Once the code is been divided into blocks. If the block includes dead codes, loops, expressions they will be further optimized for improved efficiency. Once the entire code is been analyzed for the particular block. Identify the dead codes. Identification process is done by finding out declared and unused expression, expressions whose output doesn't impact the programs execution and unreachable codes. In each and every block we are removing the dead code which helps in faster execution of code, code minimized etc. once the dead code elimination is ended the loop optimization starts up for each and every block.

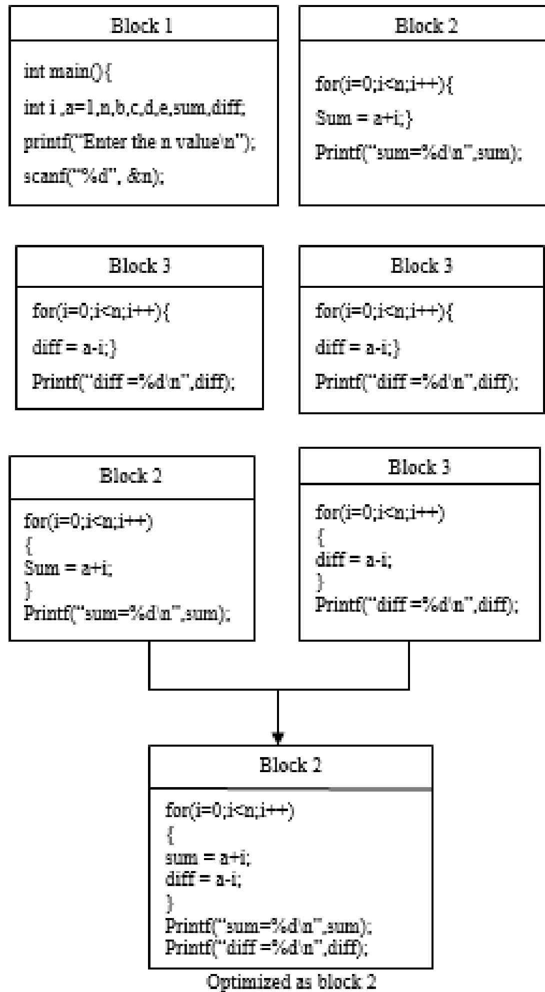
While analyzing the block check for the code which results in same output if present inside or outside the loop. Then place such codes outside the loops. If there are two blocks with two separate loops then combine such blocks into one but with some constraints applied. The loops should have same number of iterations, shouldn't have code in-between them. Such codes which satisfy the above conditions can be merged into two and proceeded further. If loop variables indexes into an array then interchange the inner loop to outer loop. Such helps in improving the locality. When a while loop is encountered replace the code by do while loop for reducing the number of jumps. When nested loops are seen apply skewing. If there are no dependencies between the loops then partition them with the many number of blocks. After the loop optimization perform common sub expression elimination. Find the sub expression in the program and compute the expression. Store the computed expression in the particular variable and replace the variable when required instead of computing the expression again and again.

D. Example

```
int main()
{
int i ,a=1,n,b,c,d,e,sum,diff;
printf("Enter the n value\n");
scanf("%d", &n);
for(i=0;i<n;i++)
{
sum = a+i;
}
Printf("sum=%d\n",sum);
for(i=0;i<n;i++)      //multiple loops
{
diff = a-i;
}
Printf("diff =%d\n",diff);
b=sum;      //dead code
if(b>10)
{
```

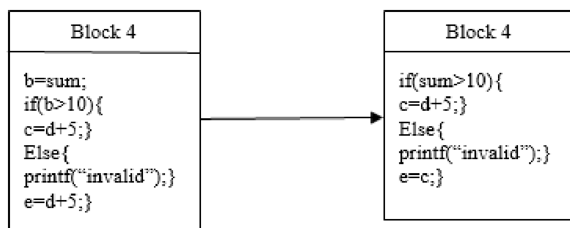
```
c=d+5;
}
Else
{
printf("invalid");
}
e=d+5;
 }        //common sub expression
```

E. Blocks



Optimized as block 2

Here in the above block 2 regenerated by combing two loops into one with the constraints applied to improve the efficiency.



In the above optimized block 4 code the dead code like b= sum; has been eliminated and the common sub expression like d+5 computation for the second time has been optimized.

3. Future implementation

Once the code has been completed developed a machine learning algorithm can be used. A pattern matching algorithm can be used. Pattern matching algorithm can be used for the each and every block. So that only the required optimization technique can be used for the particular code. This method can be used to improve the efficiency. And this method can also be added with a parallelization i.e. more than one blocks executed at the same time.

4. Conclusion

The major objective of our proposed paper is to combine the code optimization techniques. So that all the optimization techniques can be applied on a program while compiling which brings an optimized code. These optimization techniques groped system helps in increased speed execution, reduced overhead associated with loops, code improvement, less resource consumption, parallel execution of blocks, faster execution etc. so in the real world applications this code can be used very reliably. This hybrid optimization technique eliminates the dead code, invariant loops, computing same expression again and again in a program and then starts execution by dividing them into the blocks so that no other code will interpret that block of code and to maintain connectivity between these blocks i.e. to maintain flow of execution control flow graph in used. By performing all these operations finally, we can get an optimized code which can be execute faster by consuming less resource and time.

References

[1] I. Ştirb and H. Ciocârlie, "Improving performance and energy consumption with loop fusion optimization and parallelization," *2016 IEEE 17th International Symposium on Computational Intelligence and Informatics (CINTI)*, Budapest, 2016, pp. 000099-000104.
[2] Anjan Kumar Sarma, "New trends and Challenges in Source Code Optimization," in International Journal of Computer Applications, Volume 131, December 2015.
[3] Paul B. Schneck, "A Survey of Compiler Optimization Techniques," 1973.
[4] Neeraj Kumar, Saroj Hiranwal, "Improving Code Efficiency by Code Optimising Techniques," in International Research Journal of Engineering and Technology, vol. 3, no. 4, April 2016.
[5] https://en.wikipedia.org/wiki/Loop_optimization
[6] https://en.wikipedia.org/wiki/Common_subexpression_elimination
[7] https://en.wikipedia.org/wiki/Dead_code