An Optimal Code Heuristic Approach for Compiler Optimization using Graph Coloring Technique

Prateek Saraf
School of computing science and engineering
VIT UNIVERSITY

Rajnish Dashora
School of computing science and engineering
VIT UNIVERSITY

ABSTRACT
Graph theory has found its applications in various fields of computation involved in day to day life. A problem solving approach that incorporates graph theory has an added advantage of being simple and visually more comprehensible [9]. Data mining, image processing, astrology and astronomy, theoretical computer science, artificial intelligence and compiler optimization et al., every evolving field utilizes efficient algorithms involving graphs and their properties. Graph coloring has major applications in the field of compiler optimization. This paper proposes a heuristic approach of graph coloring for code optimization and hence, improvement in compiler performance and accuracy. Color-based merging of colored graphs reduces the use of temporary variables, increasing efficiency in memory utilization. A comparative analysis has been carried out in order to present the advantages of the proposed algorithm [5] [6].

Keywords
Compiler optimization, reduced graph coloring, Compiler optimization, Graph coloring, Graph traversal.

1. INTRODUCTION AND MOTIVATION
Graphs play a vital role in the optimization process phase of the compiler. There are several applications of graphs in different phases of compilers. Parse trees, Syntax trees, DAG's are useful in the initial phases where the code is analysis and prepared for synthesis. In synthesis part, flow graphs and basic blocks are generated for Data and control flow analysis and code optimization. Graphs can be used for peephole optimization and optimization of resister allocation problem in memory. Graph coloring which is an NP-complete problem can be implied to solve the register allocation problem. Through graph coloring a heuristic solution can be implemented for reducing the number of temporary variables and hence reducing the memory required for computation.

2. RELATED WORKS AND EXISTING SOLUTIONS
Earlier existing algorithms for optimizing compiler such as Generating syntax directed translation graph for 3 address codes. Graph coloring is one of the popular methods for register allocation problem. The existing graphs coloring heuristics generate the colored graph as the syntax tree populates every time coloring the whole graph which has a overhead of higher utilization of CPU. Use such algorithms do not efficiently optimizes the compilation process but sometimes becomes and overhead. Hence the proposed solution erects those flaws from the existing approaches [2].

3. CURRENT PROBLEM SCENARIO
The problem is to find how graph coloring needs to be done for optimal allocation of registers in compiler optimization technique for above disconnected graph hence to optimize the performance of compiler.

.

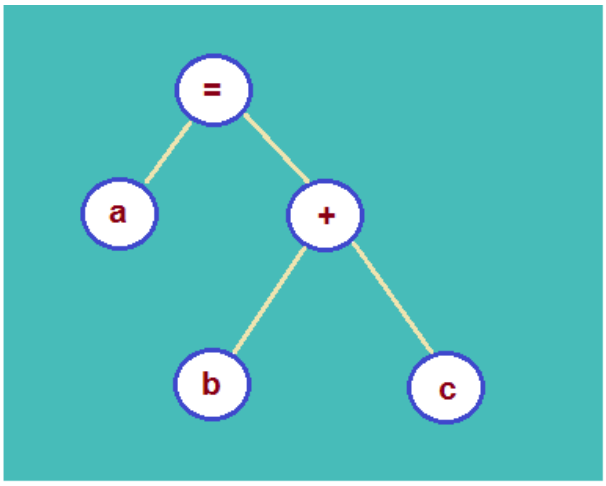

Fig 1(a) Graph G1 for a=b+c

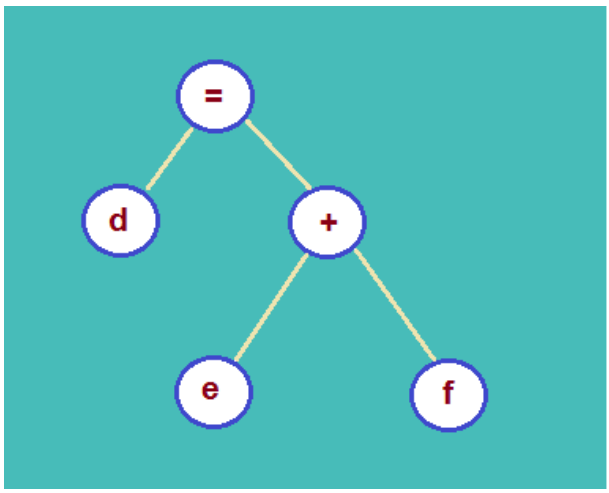

Fig 1(b) Graph G2 for d=e+f

For the above situation, graph coloring algorithm is applied – 3 times (1 for graph 1(a) + 1 for graph 1(b)) + 1 for the connected graph obtained.
Though only 3 colors are used, 3 registers required for temporary storage. Also excess use of coloring algorithm causes a heavy drop in the performance of the optimization process[10].

Color_graph(G_1,V_1,E_1){

// Coloring all the vertex such that no adjacent nodes have same colour
}
Color_graph(G_2,V_2,E_2){

// Coloring all the vertex such that no adjacent nodes have same colour
}
Color_graph(G_3,V_3,E_3){

// Coloring all the vertex such that no adjacent nodes have same colour
}

4. PROPOSED SOLUTION

Analyze the total number of vertices (say N).

No. of vertices in graph (a) = N1

No. of vertices in graph (b) = N2

Edges in graph (a) =E1

Edges in graph (b) = E2

Assuming initially colors required for coloring graph (a) be c1 and that of graph (b) will be c2. So, total colors =c1+ c2 While connecting the two disconnected graph, they must keep in mind that any vertex of graph 4(a) can connect to any of graph 4(b)

1) Color all n1 vertices with graph coloring algorithm from colors c1.

2) Color all n2 vertices with graph coloring algorithm from colors c2.

3) While (insertion of edge from Vertex v1 (a) ->vertex w1 (b))

a) if (color(vertex v1(a))==color(vertex w1(b)) choose a color from (color of alternate vertex of V(a) and fill to that vertex v1(a) similarly for w1(a) from alternate vertex in V(b) .

Else a connection is created successfully without any change in color

4) Display the final colored graph with optimal register allocation to the large extent.

5. ALGORITHM

Color_graph(G,V,E){

// Coloring all the vertex such that no adjacent nodes have same colour
}

Color_vertex(G,V,E,vert)

{

```
C[] ← color(adjacent(vert))
for(i=1;i<n;i++)
for(j=1;j<n;j++)
    if(c[i]!=c[j])
    {
        flag=i
          j=n   , i=n
    }

Color[vert]=c[flag]
```

}

Color_merge_graph(G_3,V_3,E_3)
{

Color_graph(G_1,V_1,E_1)
Color_graph(G_2,V_2,E_2)

While(vertex(V_i,V_j) in G3 where V_i in V_1 or V_2 and V_j in V_2 or V_1)
{

Color_vertex(G_3,V_3,E_3,V_i)
Color_vertex(G_3,V_3,E_3,V_j)
}

}

6. COMPARATIVE ANALYSIS AND RESULTS

For the comparative analysis of the above approach let us take and pseudo code for optimization [4] [8].

As,

a=b+c;

d=e+f;

*g=a*d-d;*

The three address code for the above will be,

Temp ← b;

Temp2←c;

Temp3←Temp1+Temp2;

a←Temp3;

Temp4←e;

Temp5←f;

Temp6←Temp4+Temp5;

d←Temp6;

Temp7←a;

Temp8←d;

*Temp9←Temp7*Temp8;*

Temp10←Temp9-Temp8;

g←Temp10;

The above 3-address code needs 10 temporary variables to execute which is not feasible.

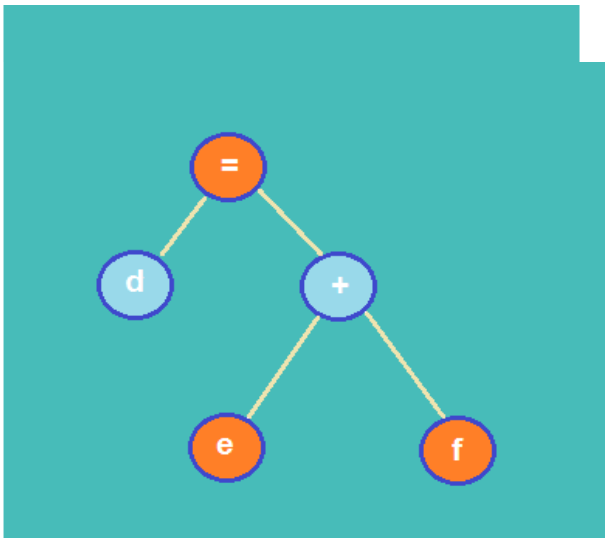

Fig 2(a) Colored Graph G1

Fig 2(b) Colored Graph G2

Using the proposed heuristics for optimization of this code,

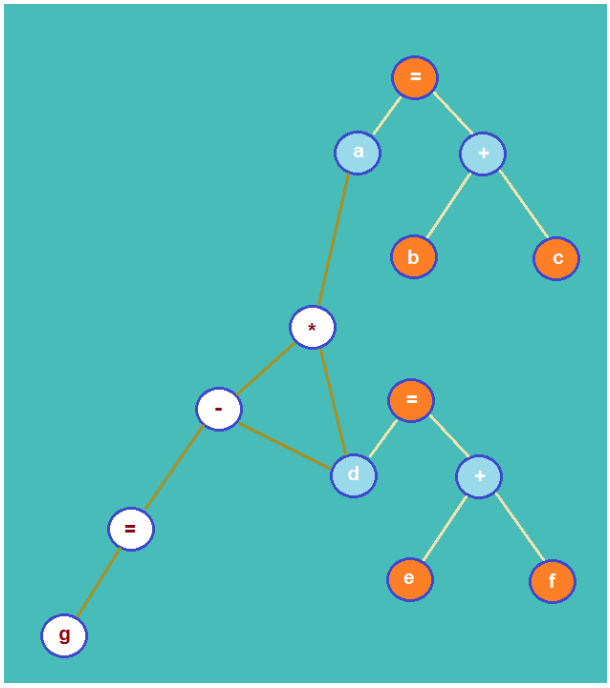

Fig 3 DAG after adding expression g=a*d-d

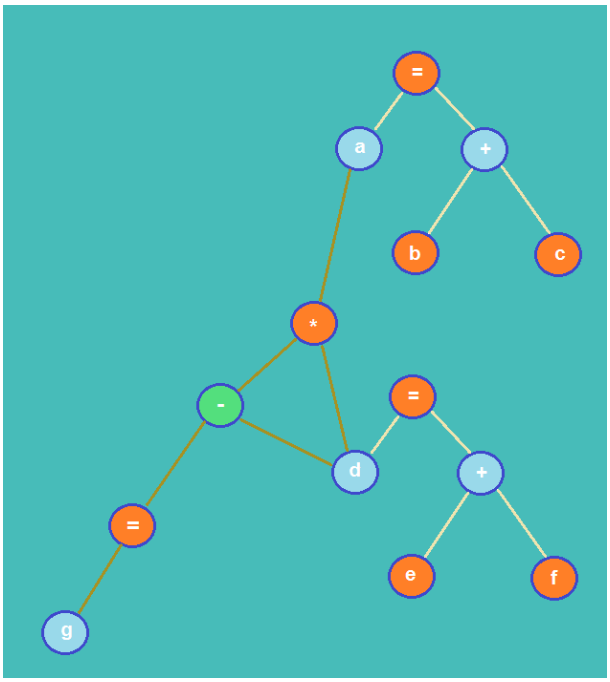

Fig 4 Colored DAG

Three address code after optimization,

Temp1←b;

Temp2←c;

a←Temp1+Temp2;

Temp1←e;

Temp2←f;

d←Temp1+Temp2;

Temp1←a;

Temp2←d;

*Temp3←Temp1*Temp2;*

g ← Temp3- Temp2;

Hence when optimized using the above approach we can analysis that the number of temporary memory locations or registers required will be equal to the number of different colors used for proper coloring of the graph which is 3 for the given scenario [7] [11].

Number of Registers Required

=Chromaticity of the Resultant DAG

Hence this analysis shows that the proposed optimal code heuristic approach for the optimization of code has significant reduction in computations required for compiler optimization. This algorithm reduces the time and space complexity for the resisters and temporary storage allocations hence, increasing the efficiency of compiler optimization techniques [1].

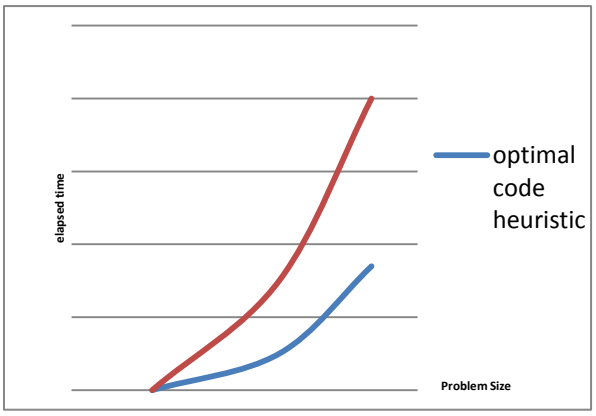

Fig 5 Efficiency of optimal code heuristic

7. CONCLUSION AND FUTURE WORKS

As the comparative analysis shows the proposed heuristic solution significantly enhances the efficiency of the code optimization hence compiler optimization. This algorithm has a significant reduction in space complexity as well, as the over head of coloring the graphs has been reduced. This enhances the performance of code and utilization memory of the system.

Future works include the application of parallelism for compiler optimization along with graph coloring which will provide an effective technique of peephole optimization for compilers. Various parallelization techniques like task decomposition and domain decomposition models are suitable for the evolution of compiler design [3].

8. ACKNOWLEDGMENTS

Our sincere thanks to all the faculty members, acquaintances and friends who helped us bring this work across.

9. REFERENCES

[1] Zhanju Cai, Alei Liang, Zhengwei Qi, Lingyan Jiang, Xiaolong Li, Haibing Guan, Ying Chen, Performance Comparison of Register Allocation Algorithms in Dynamic Binary Translation, International Conference on Knowledge and Systems Engineering, 2009.

[2] Ulrich Hirnschrott, Andreas Krall, and Bernhard Scholz, Graph Coloring vs. Optimal Register Allocation for Optimizing Compilers, Springer-Verlag Berlin Heidelberg ,2003.

[3] Istv´an Juhos, Jano I. van Hemert, Increasing the efficiency of graph coloring algorithms with a representation based on vector operations, Journal of Software, Vol1, No. 2, August, 2006.

[4] Ben A. Abderazek, Arquimedes Canedo and Masahiro Sowa, Quantitative Evaluation of Common Sub expression Elimination on Queue Machines, IEEE, 2008.

[5] Anjali Mahajan, M S Ali, Hybrid Evolutionary Algorithm for Graph Coloring Register Allocation, IEEE, 2008.

[6] David Karger, Rajeev Motwani, Madhu Sudan, Approximate Graph Coloring by Semidefinite Programming, IEEE, 1994.

[7] Monica S. Lam, Alfred V. Aho, Ravi Sethi, Jeffrey D. Ulman, Compilers: Principles, Techniques and Tools 2 Edition, Pearson 2008.

[8] Tommy R. Jensen, Graph Coloring Problems, John Wiley & Sons 1994.

[9] Narsingh Deo. Graph Theory with Applications to Engineering and Computer Science New Edition, PHI Learning 2009.

[10] Thomas H. Cormen, Charles E. Leiserson, Ronald L Rivest, Clifford Stein, Introduction To Algorithms, Mit Press (ma) 2009.

[11] Ashay Dharwadker, The Vertex Coloring Algorithm, Createspace 2011.