IOSR Journal of Computer Engineering (IOSR-JCE)
ISSN: 2278-0661, ISBN: 2278-8727, PP: 21-24
www.iosrjournals.org

An Evolutionary Algorithm based solution for Register Allocation for Embedded Systems

Mrs. Ujjwala H. Mandekar[1]
[1](J. L. Chaturvedi College of Engineering, Nagpur, India)

Abstract: *Mobile operating system consists of different types of embedded system. Generally embedded systems require optimized compilers to produce high quality codes as they have limited general purpose register set. Normally memory or registers can be used to store the results of computation of a program. But accessing a register requires less machine cycles as compared to memory access, but due to limited number, registers have to be utilized very efficiently. The main objective of this paper is to propose an efficient way to hold as many live variables as possible in registers in order to avoid expensive memory accesses. Normally register allocation problem is based on graph colouring problem. An evolutionary algorithm for graph colouring register allocation problem is used for efficient register allocation.*
Index Terms: *compilers, compiler optimization, register allocation, evolutionary algorithm, embedded systems*

I. INTRODUCTION

Register allocation is one of the most important optimization phases of any compiler. Day by day it is becoming increasingly important as the gap between processor speed and memory access time widens. The main function of this phase is to find a way to map the temporary variables used in a program into either main memory or machine registers.

Accessing a register requires less machine cycles. Hence accessing register is much faster than accessing memory. Because of this every compiler developer tries to use registers more efficiently. But due to number constraints it is not always possible. When a register is needed for a computation but all the available registers are in use, the contents of one of the used registers must be stored (spilled) into a memory location in order to free up a register. This results into additional cost,(the cost of *load* and *store* operations,) which should be avoided as much as possible. Therefore, compilers are designed in such a way that it should minimize the register requirements and the number of memory transferring instructions.

Evolutionary approach attempts to combine the utility of general-purpose programmable processors with the performance achieved by domain-specific architecture optimizations. Compilers for embedded processors must be designed in such a way that it should cope with these architectural optimizations and be able to exploit them.

This paper presents an evolutionary algorithm for graph colouring register allocation problem for embedded systems

Graph colouring maps the problem of assigning registers to live ranges in a program into the problem of assigning colours to nodes in an *interference graph.* For each procedure a register interference graph is constructed in which the nodes are symbolic registers and an edge connect two nodes if one is live at a point where other is defined. For example if the instructions are

X= a+b
Y=y-b

Here two nodes x and y are used. X is live at second statement which defines Y. Therefore in the graph there would be an edge between the nodes for X and Y. That is, as they are simultaneously live at some point and cannot occupy the same register. The colours used are the available registers. The graph colouring algorithm tries to "colour" the graph with a minimum number of colours in such a way that no two adjacent vertices are coloured with same colours.

To find an allocation from G, the compiler looks for a k-coloring of G, that is, an assignment of k colors to the nodes of G such that adjacent nodes always have distinct colors. If it is possible to choose k to match the number of machine registers, then k-coloring of G into a feasible register assignment for the underlying code is possible. Because graph coloring is NP-complete, the compiler uses a heuristic method to search for a coloring; it is not guaranteed to find a k-coloring for all k-colorable graphs. If a k-coloring is not discovered, some values are spilled; the values are kept in memory rather than in registers. Spilling one or more live ranges creates a new different interference graph. The compiler proceeds by iteratively spilling some live ranges and attempting to color the resulting new graph.

II. RELATED WORK

Register allocation has been widely addressed in literature, and many approaches have been proposed. Most of them are related to Chaitin's [6] approach, but very few address problems raised by embedded processor's architecture like support for irregular registers via register classes and register set concatenation. Although [13] proposed an approach using integer-programming supporting irregular register sets, this approach requires modeling of register constraints by inequations, making it difficult to implement in an industrial compiler. Briggs[3,4] proposes an approach to model register pairs in the interference graph. He introduced an improved coloring strategy that produces better allocations for many graphs on which Chaitin's method fails. The difference lies in the timing of spill decision.

Embedded processors are often characterized by a small number of registers. Algorithms and heuristics have been adapted in our infrastructure to limit spill when few registers are available. Ref. [17] presents a generalization of the *degree* < *K* test , called the ⟨ *p, q*⟩ test, to handle irregular register sets and register classes.

Mahajan [16] present a new hybrid evolutionary algorithm (HGR) for graph coloring register allocation problem for embedded systems as one of the most efficient algorithms with highly specialized, domain specific crossover and local search function.

III. AN EVOLUTIONARY ALGORITHM

There are seven steps in standard genetic algorithm as follosws --

Start with the population of n random individuals each with 1 bit chromosomes

1. Calculate the fitness f(x) of each individual.
2. Choose based on fitness, two individuals and call them parents. Remove the parent from the population.
3. Use a random procedure to determine whether to perform crossover. If so, refer to output of the crossover as the children. If not , simple refer to the parents as the children.
4. Mutate the children with the probability of mutatuin for each bit.
5. Put the two children into an empty set called the new generation.
6. Return to step 2 untill the new generation contain n individuals. Delete one chils at random if n is odd. Then replace the old population with new generation. Return to step 1.

To solve the graph colour problem for register allocation, the graph of registers is represented with the help of adjacency matrix representation. For example Table – 1(a) shows the matrix of the interference graphs. Table contains the value 0 and 1 where 0 represents no path between vertexes while 1 represent path between the two vertexes. Table -1(b) contains the list of adjacent vertices for each of the nodes

Table –1(a) ADJACENCY MATRIX OF THE GRAPH

Nodes	A	B	C	D	E	F
A	0	1	0	1	0	0
B	1	0	1	0	0	0
C	0	1	0	1	0	1
D	1	0	1	0	1	0
E	0	0	0	1	0	0
F	0	0	1	0	0	0

Table -1(b) NODES WITH ADJACENT VERTEXES

Nodes	Adjacent Vertices
A	B,D
B	A,C
C	B,D,F
D	A,C,F
E	D
F	C

IV.GENETIC ALGORITHM APPROACH TO SOLVE THE PROBLEM

1. Initialisation of parent population
2. Evaluation (fitness function)

3. Selection
4. Crossover/recombination
5. Mutation
6. Evaluate child and Go to step 3 until termination criteria satisfies

Initialization of parent population

• Generate the M number of solution string known as parent population
• Mostly random
• In generated chromosomes 0 represents white colour while 1 represent black colour.
• Here 4 chromosomes a, b, c and d are generated randomly with the help of a function

X 1 0 1 1 0 1
Y 0 0 1 0 1 1
Z 1 0 1 0 1 1
W 0 1 0 0 1 1

Evaluation

These four chromosomes are evaluated by a fitness function.

Fitness function

It is a function which changes from problem to problem. In this problem, for each of the vertex (Table -1(b)), its colour is checked with its adjacent vertexes. If the colour of adjacent vertexes are not same as compared to the vertex for which it is compared, fitness is increased by 1. Let us consider the first randomly generated chromosome x,

Node - A B C D E F
X -1 0 1 1 0 1

According to this randomly generated chromosome the colour of node A is 1, B is 0 and so on. Now from Table 1-(b) adjacent vertexes of node A are B and D. When the colour of Node A is compared with node B and D colour of node B is 0 but colour of node D is 1. Since all the adjacent vertexes B and D should not have the same colour of vertex a, so fitness will be 0. Similarly next node B which colour is 0 is considered and its adjacent vertexes are A and C and the colour of A and C are 1 which is not similar to the colour of B so fitness will be increased by 1. Thus when all the nodes are checked in the same way, fitness is positive only for node B and E. So total fitness for the chromosome a is 2.Similarly when the fitness function is applied to all the randomly generated chromosomes, it is found that the fitness of each chromosome is as follows……

Parent population	Fitness
X	2
Y	2
Z	4
W	3

Selection

In genetic algorithm fit solution are likely to survive and bad solution are likely to die off. So, some of the best fit chromosomes are selected from parent population according to some selection criteria (e.g. Roulette wheel selection).

Crossover/Recombination

Exchange the partial solution between the pair of selected solution with some probability value.

Mutation

Change the value of an allele of solution with some small probability value e.g. 1%. Motivation is to explore new point in the solution space.

Replace with parent population and repeat process

If after repeating the process several time if one of the solution is

H-- 0 1 0 1 0 1

Its fitness fuction after calculation will be 6. As number of nodes of interference graph is 6, this will be the best solution. Hence we can conclude that this problem can be solved using two colours.

IV. CONCLUSION

Register allocation for embedded systems is complex having irregular architectural characteristics and to meet strict requirements. In this paper, an attempt has been made for graph coloring register allocation problem for embedded systems, based on genetic algorithm. Population size of the problem should be set properly. This algorithm can also be applied to other compiler optimization problems such as instruction scheduling and phase coupling of the compiler.

REFERENCES

[1] P. Bergner, P. Dahl, D. Engebretsen, and Matthew T. O'Keefe. "Spill code minimization via interference region spilling". In SIGPLAN Conference on Programming Language Design and Implementation, pages 287–295, 1997.
[2] D. Bernstein, M. Golumbic, y. Mansour, R. Pinter, D. Goldin, H. Krawczyk, and I. Nahshon. "Spill code minimization techniques for optimizing compliers". In Proceedings of the ACM SIGPLAN 1989 Conference on Programming language design and implementation, pages 258–263. ACM Press, 1989.
[3] P. Briggs. "Register Allocation via Graph Coloring". Ph.d Thesis, Rice University, April 1992.
[4] P. Briggs, K.Cooper, and L. Torczon. "Improvements to graph coloring register allocation". ACM Transactions on Programming Languages and Systems, 16(3):428-455, May 1994.
[5] D. Brelaz, " New methods to color the vertices of a graph. Communication". ACM, vol. 22, no 4, pages 251-256, 1979.
[6] G.J. Chaitin. "Register Allocation and Spilling via Graph Coloring". Proceedings of the ACM SIGPLAN '82 Symposium on Compiler Construction, SIGPLAN Notices 17(6):98-105, June 1982.
[7] M. Chams, A. Hertz, and D. de Werra. "Some experiments with simulated annealing for coloring graphs". European Journal of Operational Research,(32): 260-266, 1997
[8] P. Galinier and J.K. Hao. "Hybrid evolutionary algorithms for graph coloring". Journal of Combinatorial Optimization,(3):379-397,1999
[9] L. George, A. Appel. "Iterated Register Coalescing". ACM Transactions on Programming Languages and Systems, 18(3):300-324, May 1996.
[10] A. Hertz, and D. de Werra. "Using Tabu search technique for coloring graphs". *Computing*,(39):345-351,1987
[11] U. Hirnschrott, A.Krall, B. Scholz. "Graph Coloring vs. Optimal Register Allocation for Optimizing Compliers". In Proceeding of the Joint Modular Languages Conference (JMLC'03), Lecture Notes in Computer Science (LNCS), Vol. 2789, pp. 202-213, Springer Press, Klagenfurt, Austria August 2003.
[12] M. S. Johnson and T. C. Miller. "Effectiveness of a machine-level, global optimizer". In *SIGPLAN '86: Proceedings of the 1986 SIGPLAN symposium on Compiler construction*, pages 99–108, New York, NY, USA, 1986. ACM Press.
[13] S. Jung, Y. Paek. "The Very Portable Optimizer for Digital Signal Processors". *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES'01),November 2001, Pages 84-92.*
[14] T. Kong, K.D. Wilken. "Precise Register Allocation for Irregular Architecture". *Proceedings of the 31st Annual ACM/IEEE International Symposium on Micro architecture (MICRO '98), November 1998, Pages 297-307.*
[15] D. Koes and S.C.Goldstein. "A progressive register allocation for irregular architectures". *In 3rd IEEE/ACM International Symposium on Code generation and optimization (CGO 2005), San Jose, CA, USA*, pages 269-280, IEEE Computer Society, 2005
[16] A. Mahajan, M.S.Ali. "Hybrid Evolutionary Algorithm for Graph Coloring Register Allocation". *Proceedings of the 2008 IEEE Congress on Evolutionary Computation (WCCI-CEC '08),Hong kong June 2008.*
[17] J. Runeson, S-O Nystrom. "Retargetable Graph-Coloring Register Allocation for Irregular Architectures". *Proceedings of the 7th International Workshop on Software and Compilers for Embedded Systems (SCOPES'03), September 2003, Pages 240-254.*
[18] B. Scholz and E. Eckstein. "Register allocation for irregular architectures". *Proceedings of Joint Conference*
[19] Anand Kumar "A novel genetic algorithm to solve map colour problem" *Proceedings of the 1st International Conference on Emerging trends Software and Compilers for Embedded in Engineering and Technology, Pages 288-291.*