ROYAL DANISH LIBR - COPENHAGEN CUL DKB (DKB)

"211455657"

LOCAL ID

TITLE

Proceedings [of the] 2. Hawaii international conference on system sciences : conference held Jan. 22-24, 1969, Honolulu, Hawaii : sponsored by Department of Electrical

AUTHOR

VOLUME

ISSUE

ARTICLE AUTHOR

Cocke, John and Miller, Raymond

ARTICLE TITLE

Some Analysis Techniques for Optimizing Computer Programs

DATE

PAGES

ISBN

ISSN

DUE DATE

BORROWER *TFW*

SUPPLIER *DKB*

PATRON, PLEASE RETURN ITEM TO:

Tufts University, Tisch Library/ILL
35 Professors Row
Medford, MA, US 02155

BORROWING LIBRARY, RETURN TO:

Royal Danish Library (DKB)
Christians Brygge 8 1219 DK
Copenhagen K,, DK 1219

SOME ANALYSIS TECHNIQUES FOR OPTIMIZING COMPUTER PROGRAMS

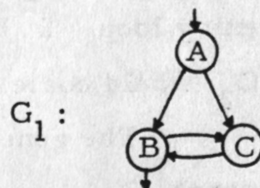
by

John Cocke and Raymond E. Miller *
IBM Watson Research Center
Yorktown Heights, New York

I. Introduction. In the process of translating a computer program written in a high level language to a machine language program, it is possible to obtain many different machine language programs each of which is "equivalent" to the original program. By including some program analysis and manipulation in the compiling process, it is often possible to obtain an improved machine language program; that is, one having fewer machine language instructions or requiring considerably less running time. An example of such an "optimization," which is not dependent upon the machine to be used, is moving an instruction out of a high frequency loop when all but the first performance of the instruction are unncessary.

In this paper, we describe several techniques for machine independent modification. In particular, we describe some systematic modifications of the flow structure of programs. We (1) assume an initial determinable flow pattern, and (2) restrict the meaning of "optimize" to circumvent any possibility of mathematical undecidability.

The model we use to represent the flow or sequencing of a program is a directed graph G, called the program graph, in which the nodes represent instructions, or blocks of instructions, and the edges represent the possible flow between blocks. Thus, in graph G_1, we denote that A is the starting point of the program by

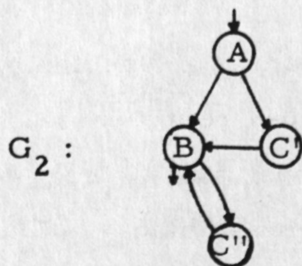


* Research partially sponsored by the Office of Naval Research under Contract No. N0014-69-C-0023.

the initial arrow to A, and by the exit arrow, that B is the end or exit from the program. Here, any possible trace through the graph from initial arrow to the exit arrow gives a possible sequence of steps in the program.

II. <u>Transformation by Node Splitting</u>. We mentioned earlier a case in which code could be moved from a high frequency portion of a program into a lower frequency portion. A simple dependency analysis, which locates operand-generation and result-uses of instructions, provides much of the information needed to determine if and where instructions can be moved. If each loop in the program (cycle in the graph) has only a single place from which it can be entered on any path from the beginning of the program, then the analysis of possible movement is particularly simple. As part of the analysis, any portion of the graph having only one entry point (whether it is a cycle or not) is successively coalesced to a single node. We consider here program graphs which contain multiple entry cycles and show how they can be transformed to equivalent program graphs having only single entry regions. This modification is accomplished by a succession of node splitting and coalescing steps.

To illustrate, consider again graph G_1. In G_1, both nodes B and C are entry points to the cycle (B, C) and no coalescing of nodes is possible. Consider G_2 in which two copies C' and C" of node C have been made. Obviously, G_2 is equivalent to G_1. In G_2, however, BC" is a single entry loop, B being the only entry node, so G_2 would coalesce.



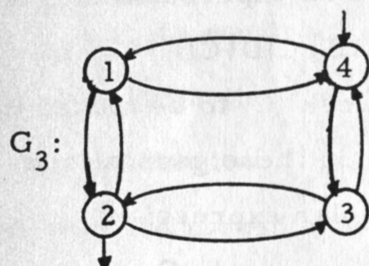
The general technique for splitting a graph G consists of two parts.

<u>Part I</u>: Obtaining a subset of nodes S, which when removed from G, together with all edges incident to S, gives a cycle-free graph. Any

set S must have the property that it contains at least one node in each cycle of G. If the cycles are known, the problem of finding an S having a minimum number of nodes is a straightforward covering problem [2]. The problem of finding sets S of minimum cardinality arises also in other contexts [1, 2].

A certain set of minimum length cycles, called prime cycles, can be shown to be the only cycles needed in considering this problem. A method based on iterative node removal and elimination of redundant terms has been found to generate just the set of prime cycles. For G_3 we obtain:

G_3:



Prime cycles for G_3

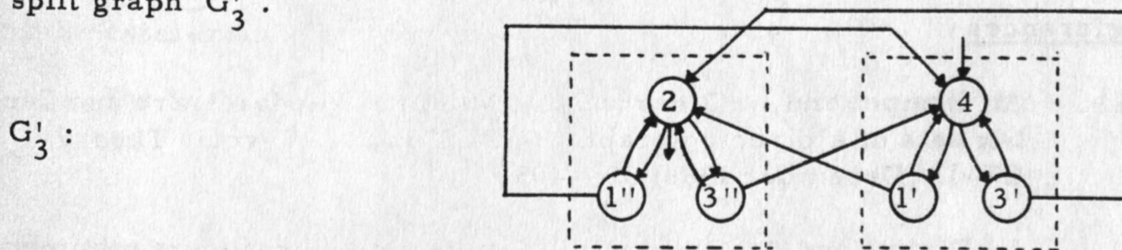
(2, 1, 2), (4, 1, 4)

(2, 3, 2), (4, 3, 4)

The solutions to the covering problem for G_3 are then $S_1 = \{1, 3\}$ and $S_2 = \{2, 4\}$.

Part II: Constructing the split graph G' from G and S. The split graph G' is formed in a rather straightforward manner by first generating an acyclic portion for each element of S and then interconnecting these portions. For our example G_3 using $S_2 = \{2, 4\}$, we obtain the split graph G'_3.

G'_3 :



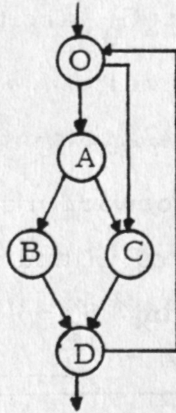
The dotted regions indicate single entry regions that can now be coalesced.

It can be shown that any such split graph G' has equivalent flow to G and when coalesced has fewer nodes than G. Thus, iteration of

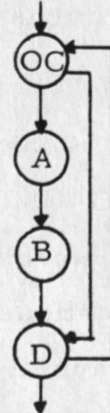
this process eventually coalesces the whole graph to a single node. Also, there is more freedom to move instructions in split graphs than in the original graphs.

III. Generalized Regular Expressions. The flow through a program graph can also be indicated by a regular expression. For example, the regular expression for G_2 is A(B + C'B)(C''B)*. Algebraic manipulation to equivalent regular expressions provides a means for obtaining modified program graphs. Regular expressions can be generalized to include parallel operation, where A | B means that block A and B can be done simultaneously. An example of such an expression derived from G_4 having B and C in parallel is [O(A(B | C)D+CD)] *. If dependency analysis allows the instructions in block C to be moved up to block O, then using this fact plus identities on these generalized regular expressions produces the simplified regular expression [OC(ABD+D)] * which gives the simplified program graph G_5.

G_4 :



G_5 :



References

[1] A. Lempel and I. Cederbaum, "Minimum feedback arc and vertex sets of a directed graph," IEEE Trans. Circuit Theory, CT-13 (December 1966) 399-403.

[2] L. Divieti and A. Grasselli, "On the determination of minimum feedback arc and vertex sets," IEEE Trans. Circuit Theory, CT-15 (March 1968) 86-89.

[3] F. E. Hohn, S. Seshu, and D. D. Aufenkamp, "The theory of nets," IRE Trans. Electronic Computers, EC-6 (September 1957) 154-161.