Proposed Method for Computing Interprocedure Slicing

Lipika Jha (1) and Dr. K.S. Patnaik (2)

(1) Department of Computer Science and engineering, Birla Institute of Technology, Mesra, Ranchi(India)
E-mail: lipika@bitmesra.ac.in

(2) Department of Computer Science and engineering, Birla Institute of Technology, Mesra, Ranchi(India)
E-mail: ktosri@gmail.com

ABSTRACT

Program slicing was originally introduced by Mark Weiser, is useful in program debugging, automatic parallelization, software maintenance, program integration etc. It is a method for automatically decomposing programs by analyzing their data flow and control flow reduces the program to a minimal form called "slice" which still produces that behavior. Interprocedure slicing is the slicing of multiprocedure program .In this paper a new method or algorithm (IP algorithm) is introduced for the interprocedure static slicing of structured programs. The most time consuming part of the interprocedure slicing methods is the computation of transitive dependences (i.e. summary edges) due to the procedure calls. Horowitz et al. [8] introduced an algorithm based on attribute grammar for computing summary edges. Reps et al. [7] and Istavan [9] defined an improved algorithm for computing summary edges representing interprocedural dependences at procedure calls. Here in this paper we discuss the improved interprocedure slicing algorithm (IP) algorithm, which is faster than previous algorithm and takes less memory space.

Keywords: control dependence, data dependence, data flow equation, system dependence graph

1- INTRODUCTION

The slice of a program with respect to program point p and variable V consists of all statements and predicates of the program that might affect the value of V at point p. A slicing criterion is a pair (p, V) where p is a program point and V is a variable which affect or affected by the value of the variable in program P. This concept was originally discussed by Mark Weiser [5]. Slicing can help a programmer to understand complicated code, can aid in debugging, software maintenance and can be used for automatic parallelization. Slicing can be classified according to whether they only account for dependencies inside a single procedure (intraprocedure slicing) or can handle slicing across boundaries (interprocedure slicing) Weiser used the directly-affects relation that is data and control dependence to compute monolithic procedure (i.e. intraprocedure slicing). Ottenstein and Ottenstein used the program dependence graphs for intraprocedure slicing. According to Ottenstein [3], the slicing problem is simply a vertex-reachability problem, and thus slices may be computed in linear time. For interprocedure slicing, the slicing criteria is extended to the called procedure and calling procedure. Horowitz, Reps and Binkley hence forth referred as HRB extended [8] this program dependence graph (PDG) to system dependence graph (SDG) for computing interprocedure slicing. Interprocedure slicing is generating a slice of an entire program, where the slice crosses the boundaries of procedure calls.

This paper is concerned with the problem of interprocedure slicing. In the concept of system dependence graph (SDG) [3] was introduced to construct the control-flow and data-flow representations of programs. The most time-consuming part of this approach is the computation of the transitive dependences due to the procedure calls (interprocedural dependences).Reps et al. [7] and Istavan [9] defined an improved algorithm for computing summary edges representing interproceural dependences at procedure calls. Here in this paper we discuss the improved interprocedure slicing algorithm (IP) algorithm, which is faster than previous algorithm and takes less memory space. The IP algorithm therefore is optimal.

This paper is organized as follows. Section 2 discusses about the techniques to compute interprocedure slicing. Section 3 elaborates the proposed algorithm (IP algorithm) .Section 4 discusses the comparative analysis of different algorithms and its complexity followed by conclusion and references.

2- BACKGROUND

2-1 DATA FLOW EQUATION

Weiser's definition [2] of program slicing is based on iterative solution of data flow equations. He extended his work for interprocedure slicing [4] which involves three separate tasks:

- First, interprocedure summary information is computed. For each procedure P, a set MOD(P) and USE(P) is computed
- The effect of call-statements on the sets of relevant variables and statements are computed using the summary information. A call to procedure P is treated as a conditional assignment statement '**if <**SomePredicate> **then** MOD(P) := USE(P) where actual parameters are substituted for formal parameters.
- The third part is to generate new slicing criteria for called and calling procedure with respect to which intraprocedure slices are computed. For each procedure P, new criteria are generated for

(i) procedures Q called by P-the criteria consists of all pairs (n_Q, V_Q), where n_Q is the last statement of Q and V_Q is the set of relevant variables in P in the scope of Q (formals are substituted for actual) which is denoted by DOWN(n_Q, V_Q),

(ii) procedures R that call P-the new criteria consist of all pairs (N_R, V_R) such that N_R is a call to P in R, and V_R is the set of relevant variables at the first statement of P that is in the scope of R (actual are substituted for formal)which is denoted by UP(N_R, V_R).

Program Example

```
                                    Procedure Add (a, b)
 begin                              begin
(1) read(n);                        (11) a := a + b;
(2) i := 1;                          end
(3) sum := 0;
(4) product := 1;                    procedure Multiply(c, d)
(5) while i<=n do                          begin
begin                               (12) j := 1;
(6) Add(sum, i);                     (13) k := 0;
(7) Multiply(product,i);           (14) while j< d do
(8) Add(i, 1)                               begin
end;                                       (15) Add(k, c);
(9) write(sum);                             (16) Add(j, 1);
(10) write(product);                       end;
end                                   (17) c := k
                                        end
```

Figure 1 Example of a multi-procedure program

According to the Weiser's terminology to compute interprocedure slicing:

(i) Compute MOD() and USE() for each procedure of P using data flow analysis

(ii) The extended criterion for Q is

$$(n_e^Q, ROUT(i)_{F \to A} \cap SCOPE_Q) \quad (1)$$

Where n_e^Q is the last statement in Q, $F \to A$ means substitute formal for actual parameters, $SCOPE_Q$ is the set of variables accessible from the scope of Q, and

$$ROUT(i) = \cup_{j \in SUCC(i)} R_C(j)$$
(2)

(iii) Third step is to create new criteria for all procedures called by P, DOWN(P) and all the procedures who called P the slicing criteria denoted as UP(P).

The problem with Weiser's method is the generation of too many criteria which are extraneous to the program. Weiser's algorithm does not produce as precise slice because transitive closure fails to account for the calling context problem.

2-2 DEPENDENCE GRAPH

Weiser,s UP/DOWN relations [3], for computing interprocedure slicing generates the unprecise slice He uses iterative data-flow control flow which is not suited to the PDG/SDG in which slices can be taken only where a variable is defined or referenced. Horwitz, Reps, and Binkley (HRB) [7] used system dependence graph for computing interprocedure static slices.

A SDG contains a program dependence graph for the main program, and for each procedure. Using this model, data dependences between procedures are limited to dependences from actual-in vertices to formal-in vertices and from formal-out vertices to actual-out vertices. Connecting procedure dependence graphs to form a system dependence graph is straightforward, involving the addition of three new kinds of edges:
(1) a call edge is added from each call-site vertex to the corresponding procedure-entry vertex;
(2) a parameter-in edge is added from each actual-in vertex at a call site to the corresponding formal-in vertex in the called procedure;
(3) a parameter-out edge is added from each formal-out vertex in the called procedure to the corresponding actual-out vertex at the call site. (Call edges are a new kind of control dependence edge; parameter-in and parameter-out edges are new kinds of data dependence edges.)

Another advantage of this model is that flow dependences can be computed in the usual way, using data-flow analysis on the procedure's control-flow graph. The control-flow graph for a procedure includes nodes analogous to the actual-in, actual-out, formal-in and formal-out vertices of the procedure dependence graph. A procedure's control-flow graph starts with a sequence of assignments that copy values from call temporaries to formal parameters and ends with a sequence of assignments that copy values from formal parameters to return temporaries. Each call statement within the procedure is represented in the procedure's control-flow graph by a sequence of assignments that copy values from actual parameters to call temporaries, followed by a sequence of assignments that copy values from return temporaries to actual parameters. We assume that all actual parameters are copied into the call temporaries and retrieved from the return temporaries.

The Horowitz slicing algorithm consists of the three steps:
1. Create *System Dependence Graph* (SDG), a graph representation for multi-procedure programs.
2. The computation of interprocedural summary information that is to create *summary edges* between the vertices and show the transitive dependence relation.
3. A *two-pass algorithm* for extracting interprocedure slices from an SDG.

The two pass for extracting interprocedure slicing are:
Pass 1 follows flow edges, control edges, call edges, and parameter-in edges, but do not follow def-order edges or parameter-out edges.
Pass 2 follows flow edges, control edges, and parameter-out edges, but does not follow def-order edges, call edges, or parameter-in edges.
The system dependence graph and its associated algorithms solve the entire earlier problem all at once. This is why it is superior representation of the program. The HRB algorithm computes the precise slice but it needs the knowledge of attribute grammar for finding the transitive dependence edges and it is very complex and lengthy.

2-3 AL ALGORITHM

The AL algorithm [1] also uses the SDG representation for slicing. This improves upon the HRB algorithm which in the worst case traverses an edge twice. The AL algorithm maintains a three valued tag that helps in distinguishing the calling context as well as identifying the vertices in the slice. Like HRB algorithm it maintains a Worklist of vertices that have been marked so far and as vertices are traversed they are tagged (T, U, S). Contrast this with the HRB algorithm's use of tags with two values: marked and unmarked .In the beginning of the AL algorithm all vertices in the slicing criterion are placed in the Worklist and are assigned the tag T. All other vertices are assigned the tag U. At the end of the algorithm all vertices whose tags (T,S) are in the slice. The traversal requires picking each edge $e = w \rightarrow v$ in the SDG corresponding to a vertex v in the Worklist and deciding:

1. Whether w should be put in the Worklist and, if so,
2. What the value of its tag should be.

When traversing Intra edges (i.e. within the PDG) the tag at the target vertex of the edge is propagated to the vertex at the source. This is not however the case when From-To and To-From edges are traversed. A From-To edge (called to caller) is not traversed if the tag of the target vertex is not T. When it is traversed the tag T is propagated to the source vertex. A To-From edge (caller to called) is always traversed. Irrespective of the tag of its target vertex a tag is propagated to its source vertex. The source vertex is put in the Worklist only if its tag changes.

3- IP ALGORITHM

Firstly, the weiser's data flow equation was used for computing interprocedure slicing. Then Horowitz introduced the concept of SDG, summary edges and 2 pass algorithm. AL uses the SDG, summary edges and one pass algorithm. For computing summary edges lots of time is required which overall reduces the speed of slicing. For increasing the speed of slicing a no. of algorithm [7], [9] was introduced for computing summary edges. Till now lots of work has been done to reduce the time to compute summary edges. The algorithm which uses the SDG requires computing summary edges. In IP algorithm we don't require the SDG for computing slice. Here we will have the PDG for each procedure and the PDG will be connected with each other using call-entry and call-site vertex. Like HRB algorithm [1] it also maintains a worklist of vertices that have been marked so far and as vertices are traversed they are tagged. The IP algorithm uses the set {T, I, B} of three tags, T for visited vertex, I for sliced vertex and B for unvisited vertex.IP algorithm for interprocedure slicing is given in figure 2.

```
Procedure IP_slicing_algorithm (G,S)
declare
    G = system dependence graph
    Set = set of sliced vertices
    Worklist = set of vertices
begin
    Worklist=v
    mark tags of  all vertices to B
    While Worklist ≠ ø
        select and remove a vertex v from Worklist
        If v ≠ T
        mark the tag T to v
        If v =procedure and  USE(P)=MOD(v) then
            mark the tag  I to v
            put v in Set
            for each vertex w such that edge $e = w \rightarrow v$ in G
            Worklist=w
            Add slicing vertex a and called proceure p to Worklist
        else if v ≠ procedure then
            mark the tag I to v
            put v in Set
            for each vertex w such that edge $e = w \rightarrow v$ in G
            Worklist=w
        end if
    end while
end
```

Figure 2 IP Algorithm

In IP algorithm we create the SDG without including the parameter_in, parameter_out vertex and parameter_in and parameter_out edges. Here we include the call site vertex and entry vertex.

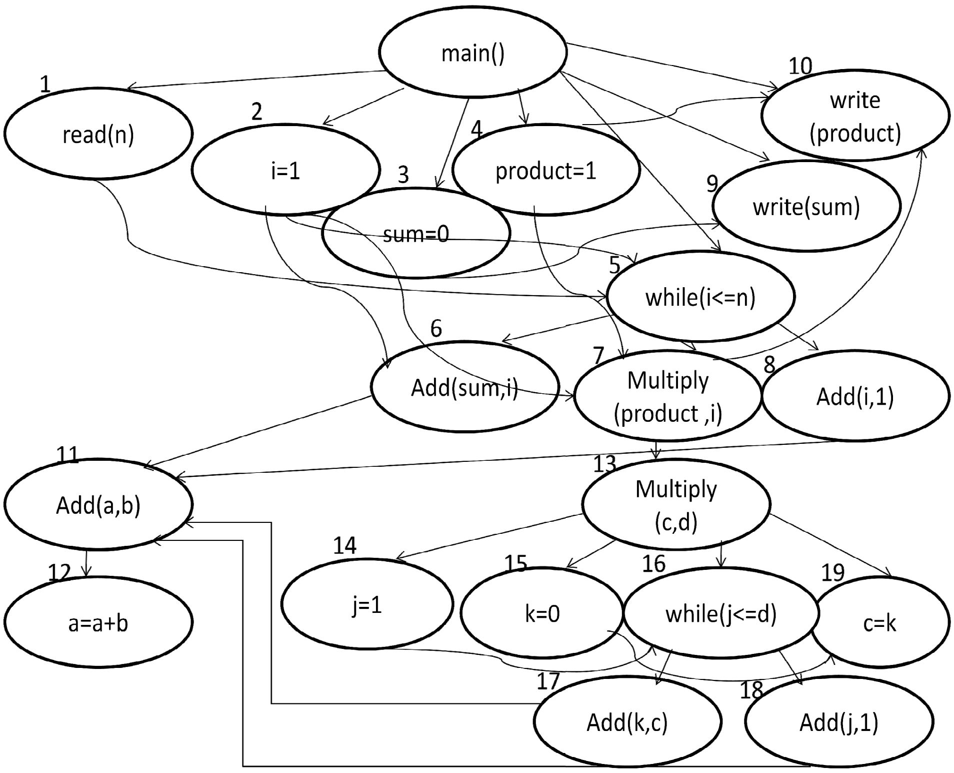

Figure 3 System Dependence graph of Figure 1 based on slicing criteria(10,product)

MOD(P)=i,n,product USE(P)=i,n,product
MOD(Add(sum,i))=a   USE(Add(sum,i))=a,b
MOD(Multiply(product,i)=j,k,c
USE(Multiply(product,i)=j,k,c,d
MOD(Add(i,1))=a   USE(Add(i,1))=i
MOD(Add(sum,i))=a   USE(Add(sum,i))=a,b
MOD(Add(k,c))=k   USE(Add(k,c))=k,c
MOD(Add(j,1))=j   USE(Add(j,1))=j

Table 1 Application of IP algorithm for slicing the SDG of figure 1 at vertex set {10}.The first column shows the Worklist. At each iteration the first element form the list is selected that is denoted by V column. The marked columns specify the marked tag of the vertices. Next column specify the inedge vertices of the respective vertices. The last column gives the number of edges traversed in the previous iteration.

Worklist	Vertices	Marked	Set	Inedges vertices	No. of Edges Traverse
10	10	T,I	10	7,4	-
7,4	7	T,I	10,7	5,4,2	2
4,5, 2,13 ,19	4	T,I	10,7,4	0	5
5,2, 13,19	5	T,I	10,7,4,5	6,7,8,1,2	0
2,13 ,19, 6,7, 8,1, 2,	1	T,I	10,7,4,5,2	0	5
13,19,6, 7,8, 1	13	T,I	10,7,4,5,2,13	7	0
19,6 ,7,8, 1	19	T,I	10,7,4,5,2,13,19	13,15,17	1
6,7, 8,1, 15,17	6	T	10,7,4,5, 2,13,19	-	3
8,1, 15,17	8	T,I	10,7,4,5,2,13,19 ,8	5	0
1,15 ,17, 11,12	1	T,I	10,7,4,5,2,13,19 ,8,1	-	2
15,17,11	15	T,I	10,7,4,5,2,13,19 ,8,1,15	13	-

,12					
17,11,12	17	T,I	10,7,4,5,2,13,19,8,1,15,17	16,15	1
11,12,16,15	11	T,I	10,7,4,5,2,13,19,8,1,15,17,11	6,8	2
12,16,15	12	T,I	10,7,4,5,2,13,19,8,1,15,17,11,12	11	2
16,15	16	T,I	10,7,4,5,2,13,19,8,1,15,17,11,12,16	14,13	1
15,18,14	15	T,I	10,7,4,5,2,13,19,8,1,15,17,11,12,16,15	13	2
18,14	18	T,I	10,7,4,5,2,13,19,8,1,15,17,11,12,16,15,18	14,16	1
14	14	T,I	10,7,4,5,2,13,19,8,1,15,17,11,12,16,15,18,14	13	2
-	-	-	10,7,4,5,2,13,19,8,1,15,17,11,12,16,15,18,14	-	1

Total edges traversed : 30

The vertex at the Set column contains all the sliced vertex of a program 2.

4- COMPARATIVE ANALYSIS BETWEEN HRB, AL AND IP ALGORITHM

For computing interprocedure slicing the HRB algorithm uses two pass algorithms and the AL algorithm uses the one pass algorithm but it requires lot of time to compute summary edges. Our algorithm doesn't require any summary edges it compute slices based on the intra_edges. Depending on the intra_edges of the vertices, the vertices are traversed. For figure 3 the number of edges traversed by each of this algorithm is:

HRB: 78 edges traversed
AL : 50 edges traversed
IP : 30 edges traversed

From the above data we can see that number of edges traversed in case of IP algorithm is minimum. But it takes extra effort to check each vertices that it is a call site vertex or not and to compute MOD() and USE() for each procedure.

Here below we have taken two examples and compute the total number of edges traversed for computing interprocedure slicing using the IP algorithm.

Example1:

```
procedure Main                  procedure A(a, b)            procedure
Increment(z)
 begin                           begin                        (9) call
Add(z,1);
    (1)sum=0;                    (6) call Add(x,y);          end
    (2)i=1;                        (7) call Increment(y);
    (3)while(i<11)              end
    (4) call A(sum,i);           procedure Add(a,b)
    (5)print(sum);                 (8)  a=a+b;
end                              end
```

Figure 4 Program to display the value of sum. The slicing criteria is (4, sum)

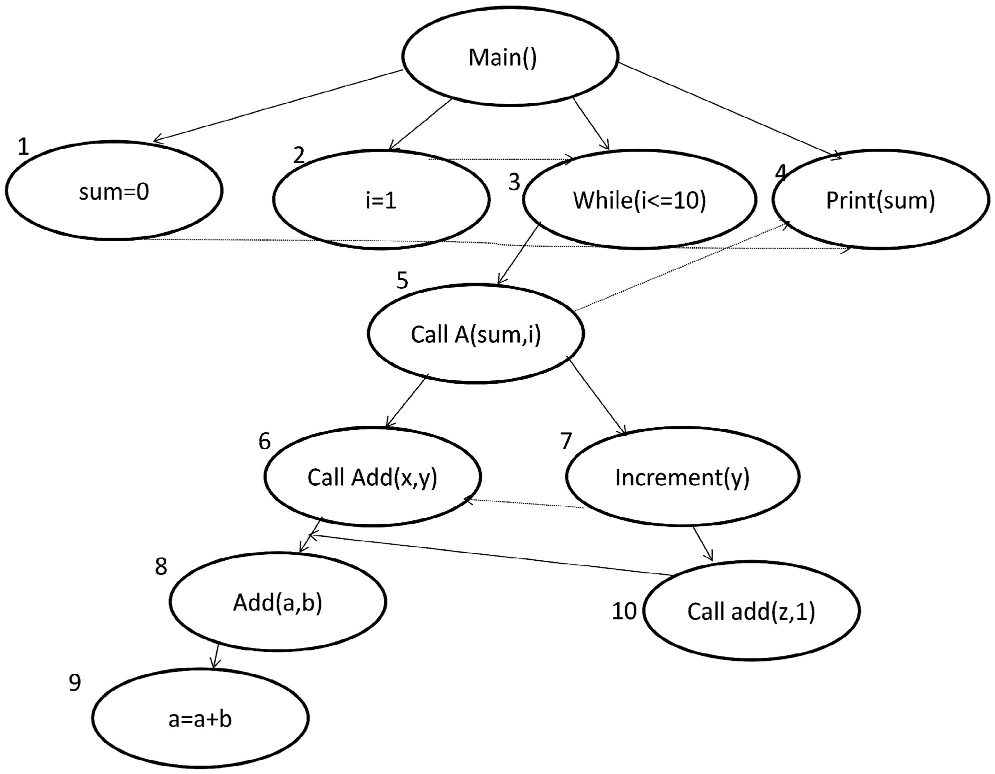

Figure 5 System Dependence graph of figure 4 based on slicing criteria (4, sum)

Table 2 Table for computing total number of edges traversed during slicing in figure 5

Worklist	Vertices	Marked	Set	Inedges vertices	No. of Edges
4	4	T,I	4	1,5	2
1,5	1		4,1	-	
5	5		4,1,5	3	1
3,7	3	T,I	4,1,5,3	2	1
7,2	7		4,1,5,3,7	5	1
2,10	2		4,1,5,3,7,2		
10	10		4,1,5,3,7,2,10		
8	8		4,1,5,3,7,2,10,8	6	1
6,9	6		4,1,5,3,7,2,10,8		
9	9		4,1,5,3,7,2,10,8,9		

Total edges traversed : 6

The vertex at the Set column contains all the sliced vertex of program in Example 1.

Example 2:

```
prcocedure Main              procedure P1(x,y)          procedure
P2(xy)
begin                        begin                      begin
(1)a=1;                        (5) if(y==0)               (8) if(xy==0)
(2)b=0;                        (6)    call P2(x);         (9)    call
P2(xy);
(3)call P1(a,b);               (7)y=y+1;                  (10)
xy=xy+1;
(4)z=b;                      end                        end
end
```

Figure 6 Program to display the value of sum. The slicing criteria is (4, sum)

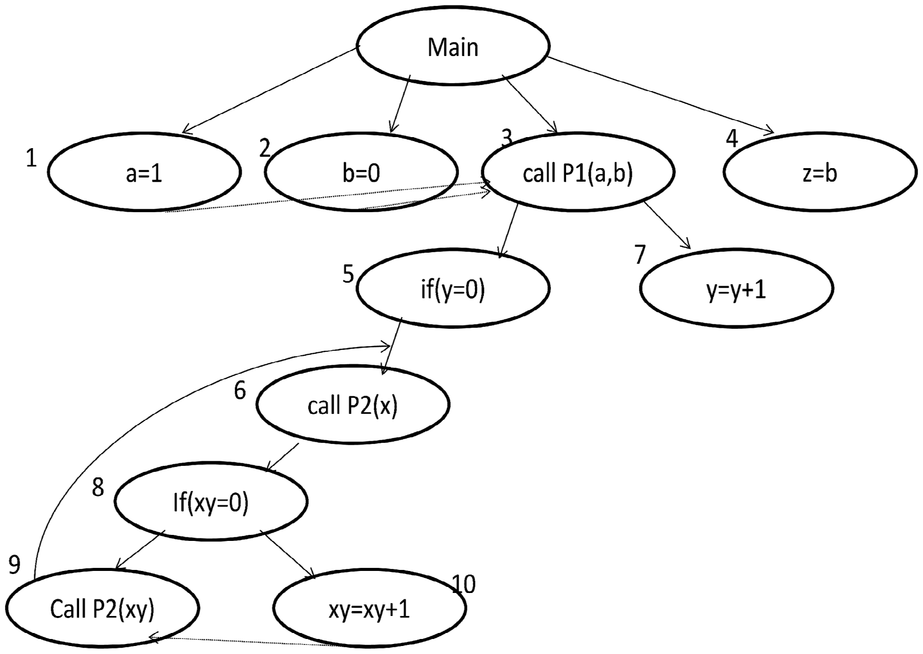

Figure 7: System Dependence graph of figure 6 based on slicing criteria(7,y)

Table 3 Table for computing total number of edges traversed during slicing in figure 7

Worklist	Vertices	Marked	Set	Inedges vertices	No. of Edges
7	7	T,I	7	3	1
3	3	T,I	7,3	2,1	2
2	2	T,I	7,3,2		
1	1	T,I	7,3,2,1		

Total edges traversed : 03

The vertex at the Set column contains all the sliced vertex of a program 2.

From the above two example we can conclude that using the IP algorithm the no. of edges traversed for computing slicing is less. Whereas in case of HRB and AL algorithm we have to first create SDG with detail of all parameter edges and for computing slicing we have to traverse from some of these parameter edges. So the number of edges will be larger in the both HRB and AL algorithm.

The space complexity of HRB and AL will be larger because we require a larger space to store the whole SDG. Whereas in case of IP algorithm we can separately store the pdg and connect the pdg with call procedure

The complexity of computing an interprocedure slice using SDG may be separated into the complexity of constructing the SDG and that of traversing this graph to identify statements in a slice. Horwitz [7] have analyzed that the cost of constructing SDG is polynomial in various parameters of the system and that the cost of traversing the SDG is bounded by the size of the SDG.A more precise complexity of the traversal algorithm may be derived as a function of the number of edges in the final slice. Let E be the number of edges in the SDG of the slice with respect to vertex set S. The IP algorithm contains less number of edges in the slice and it traverses edges at most one time.

5- CONCLUSION

This paper presents an algorithm that improves upon the interprocedure slicing algorithm presented by Horwitz, Reps, and Binkley [7] and AL (1) algorithm. Our algorithm has the same order of complexity but with an improved constant. Instead of the two traversal of edges performed by HRB algorithm, our algorithm may be implemented to perform a maximum of one traversal one per edge. Here the SDG we have created will not contain parameter_in, parameteter-out edges and vertices. This paper has been presented with a view to reduce the complexity to compute the interprocedure slicing .Weiser' s method to compute interprocedure slicing [3] was easy but it also extract irrelevant statement. Horowitz method [7] was precise slice but was very complex and time consuming. Here in this paper we have proposed a new method for computing interprocedure slice but it has not been implemented anywhere yet. The algorithm is not tested using any testing pattern. Testing part we have planned as our future work.

ACKNOWLEDGEMENT

I wish to convey my sincere gratitude and appreciation to each and every person who helped me in writing this paper. I am grateful to my institution, Birla Institute of Technology and my colleagues. I would especially like to thank Dr. K. S. Patnaik, my guide for his advice and guidance.

REFRENCES

[1] A. Lakhotia, "Improved interprocedural slicing algorithm" The Center for Advanced computer studies, University of Southwestern Louisiana, Lafayette, LA,1992.

[2] D. Binkley and K. B. Gallagher, "Program Slicing," Advances in Computers, Vol. 43,No.9 pp. 1-50,1996.

[3] F. Tip, “A Survey of Program Slicing Techniques,” Journal of Programming Languages, Vol. 3, No. 3, 1995, pp. 121-189.

[4] K. B. Gallagher, Computer Science Department University of Durham South Road Durham DH1 3LE “Some Notes on Interprocedural Program Slicing,” 4th IEEE International Workshop on Source Code Analysis and Manipulation: SCAM-4. Chicago, Illinois, 15-16 September 2004.

[5] M. Weiser, “Program Slicing,” IEEE Transactions on Software Engineering, Vol.SE-10 16, No. 4, pp. 352-357,1984.

[6] E. MYERS, “A precise inter-procedural data flow algorithm.” In Conference Record of the Eighth ACM Symposium on Principles of Programming Languages (Williamsburg, Va., Jan. 26-28, 1981).ACM, New York, pp. 219-230,1981.

[7] T. Reps, S.M. Horwitz, and G. Rosay, “Speeding up slicing” ACM SIGSOFT Engineering notes 19(5) 11-20 December, 1994.

[8] S. Horwitz, T. Reps, and D. Binkley. “Interprocedural slicing using dependence graphs”. ACM transactions on Programming Languages and Systems, 12(1):35–46, January 1990.

[9] T. Gyimothy and I. Forgacs. “An Efficient Interprocedural Slicing method for Large Programs”TR-96-106, January 1996.