Information and Management Sciences
Volume 16, Number 2, pp. 35-46, 2005

A Parallel Algorithm for Static Program Slicing

P. K. Mishra
Birla Institute of Technology
India

Abstract

Program slicing is the process of deleting statements in a program that do not affect a given set of variables at a chosen point in the program. In this paper the parallel slicing algorithm is introduced. It is shown how the control flow graph of the program to be sliced is converted into a network of concurrent processes, thereby producing a parallel version of Weiser's original static slicing algorithm. Keywords:

Keywords: Parallel Algorithm, Program Simplification, Slicing.

1. Introduction

A slice of program p is constructed with respect to a slicing criterion (V, n), by deleting statements from p which have no effect on any of the variables [2] in V as execution reaches line n. A slice is thus (typically) simpler than the original program. Weiser's original algorithm [25, 27] has been modified by several authors, who have improved upon the accuracy and speed of slice construction [21,12, 15, 22,11]. The applications of slicing include program comprehension [26], program maintenance and debugging [5, 16] program testing [18] the calculation of cohesion metrics [20] and program integration [9]. These applications rely on the way in which slicing allows for program simplification, by extracting a thread of the original program's computation.

Weiser's original approach solved the static slicing problem. Other authors have extended slicing to the dynamic [2, 6, 13,14] and quasi-static paradigms [24, 23]. The algorithm represented in this paper also solves the problem of static program slicing. It is essentially a parallel version of Weiser's and is defined in terms of the control flow graph (CFG)[7]. The CFG is compiled into a network of concurrent processes similar to those defined in [1]. Each process communicates with the others along the arcs of the CFG.

Received June 2004; Revised November 2004; Accepted January 2005.
Supported by Birla Institute of Technology, India.

The messages include sets of variables are closely related to Weiser's relevant variables [27]. To our knowledge, the algorithm introduced here is the first to exploit concurrent execution in program slice construction.

The rest of this paper is organized as follows: Section 2 describes the parallel algorithm, a formal proof of which can be found in [17]. Section 3 contains a detailed example, showing how it constructs a slice and Section 4 describes some other interesting and potentially useful properties of the parallel slicing algorithm over and above slice construction.

2. The Algorithm

The slice of the original program is created as follows:

(1) The original program is converted into a CFG.
(2) The CFG is compiled into a network of concurrent processes.
(3) The network of concurrent processes is executed. The nodes of the CFG that correspond to statements to be kept in the final slice are obtained from the resulting output.

2.1. Constructing a process network

From the program to be sliced, a process network is constructed. The topology of the network is obtained directly from the inverse of the program CFG, called the Reverse Control Flow Graph (RCFG) [3], with one process for each node in the CFG and with communication channels corresponding to the arcs of the RCFG.

As is the convention arcs entering a node i, represent inputs to process i, and arcs leaving represents output from processes, i. When a process outputs a message, it shall mean that the message is output on all output channels.

2.2 Process behaviour

Each process sends and receives message that are sets of variables names and node identifiers. The behaviour of each process, i, depends precisely on the following information, derived directly from the CFG of the program being sliced:

i	:	The identifier of the corresponding node of the CFG.
REF(i)	:	The set of variables referenced by i.
DEF(i)	:	The set of variables defined by i.
C(i)	:	The set of nodes controlled by i

Processes with more than one input in the RCFG correspond to predicate nodes. The present paper is concerned with side-effect free language, so all such processes will have DEF(i) = ϕ. Conversely, processes with only one input do not correspond to predicate nodes, and therefore, by definition, they control no other nodes and so have $C(i) = \phi$. The behaviour of each process, i, can be defined in CSP style notation [8] as follows:

P(i)=?S → (if S ∩ (DEF(i) ∪ C(i)) ≠ ϕ
then ! ((S \DEF(i)) ∪ REF(i) ∪ {i})
else !S);
P(i)

If the input, S, to process i, has elements in common with the defined variables of i or with the controlled nodes of i then the process, i, outputs the set consisting of:

(1) all its input variables (elements of S) that it does not define,
(2) all variables that it references,
(3) its node identifier, i.

On the other hand, if S has no elements in common with the defined variables or controlled nodes of i then the process i merely outputs S. The process i then repeat this action, waiting for the next input message.

2.3. Initiating network communication

In order to construct a slice for the criterion (V, n), network communication is initiated by outputting the message V from process n.

2.4. Constructing the slice

For any process the node i should be included in the final slice if and only if i has output its node identifier i. The slice of a program computed by this algorithm can be found by including the set of nodes whose identifiers are input to the entry node of the

CFG, because the entry node is reachable via every node in the RCFG and thus message output by all nodes will eventually reach the entry process.

Once the set of nodes to be included in the final slice is obtained using the parallel slicing algorithm, the problem of converting this information back into the syntax of the original program arises. The solution is well known [25,10] and is therefore not discussed further in this paper.

Section 2.2 above, should be thought of as a "specification" of process behaviour rather than an "implementation". The important aspect of the definition is that each process should be thought of as a function from the union of all its inputs to the union of all its outputs [17].

In a valid implementation of the parallel slicing algorithm [17], to ensure termination, process should, in fact, not output the same message more than once as in the case in the following example execution.

```
Step1: a=0
Step2: while (s<t) {
Step3: if (t==4)
Step4: c = t;
Step5: s=2;
Step6: c=t+7;
Step7: t=a+4;
}
```

Figure 1. The program to be sliced.

3. A worked example

Let the slicing criterion be ({c},7) and the program to be sliced be the one shown in Figure 1.

The RCFG of the program to be sliced is shown in the Figure 2.

The process network obtained from the RCFG is shown in the Figure 3. A slice is to be constructed for the criterion ({c},7), so process communication is initiated by outputting {c} from node 7.

To show the progression of the state of the system, we label the arcs in the RCFG with the message communicated by the relevant process during execution. New message communicated at each stage are labelled in bold type face.

Process are drawn like this:

Where C is the set of controlled nodes, DEF is the set of defined variables and REF is the set of referenced variables of each processes. After receiving {c} through its input channel, process six outputs {t,6}. Processes five, four and three all eventually receive {t,6} which they simply output because {t,6} is disjoint from the defined variables of these processes. The resulting state is shown in Figure 4.

When {t,6} is input to process two, it causes process two to output {s,t,2,6} to processes one and seven. This is an instance of a process responding to an input containing the identifier of a node that it controls. Process two will therefore output a message including its own node identifier, representing the fact that node two will also be included in the final slice.

The resulting state is shown in the Figure5.

The message {s,t,2,6} passes through process one to the ENTRY node. On receiving {s,t,2,6}, process seven outputs the message {s,a,2,6,7} because it defines t. This output message passes unaffected through process six.

When process five receives {s,a,2,7} it outputs {a,2,5,7}.

The resulting state is shown in Figure 6.

Continuing process communication passes the extra message in the network to all reachable nodes, but causes no new message to be introduced into the system. Finally the network terminates in the state shown in Figure 7.

From the original state of the network the slice of the original program is constructed by including those statements and predicates whose node identifiers have reached the ENTRY node.

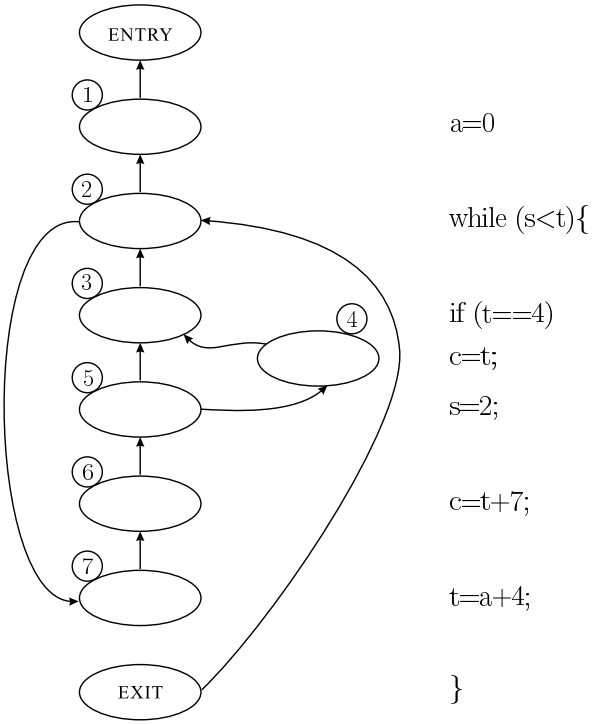

Figure 2. The RCFG obtained from the initial program.

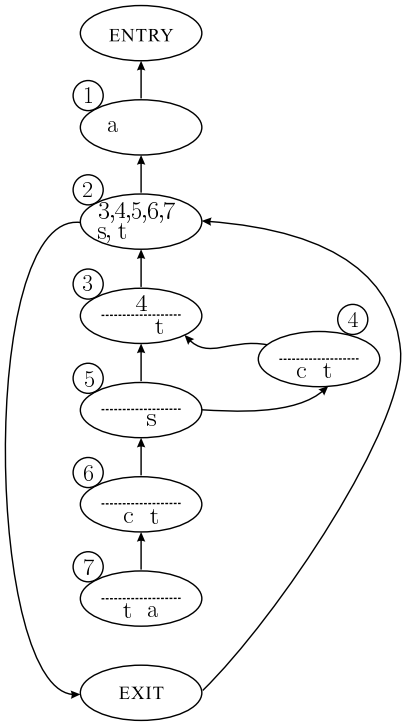

Figure 3. Initial state of the process network.

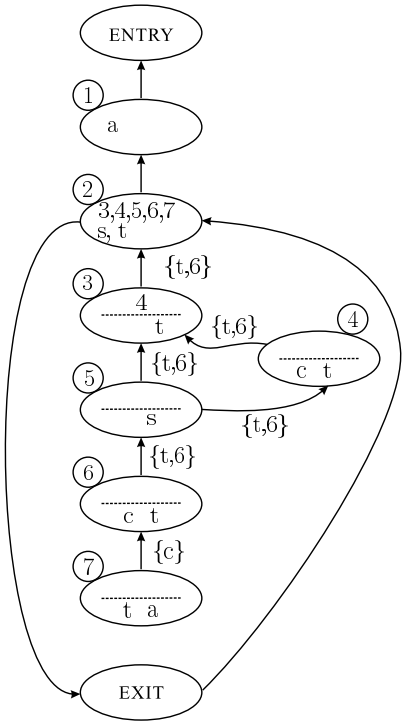

Figure 4. The state just before process two output in first message.

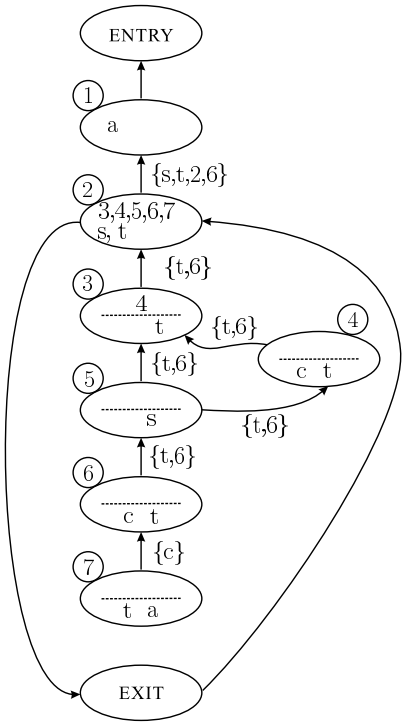

Figure 5. The state just after process has two output in first message.

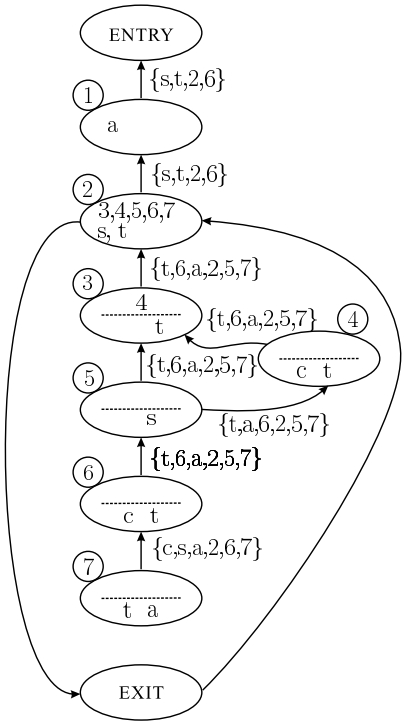

Figure 6. The state just after one move pass and loop.

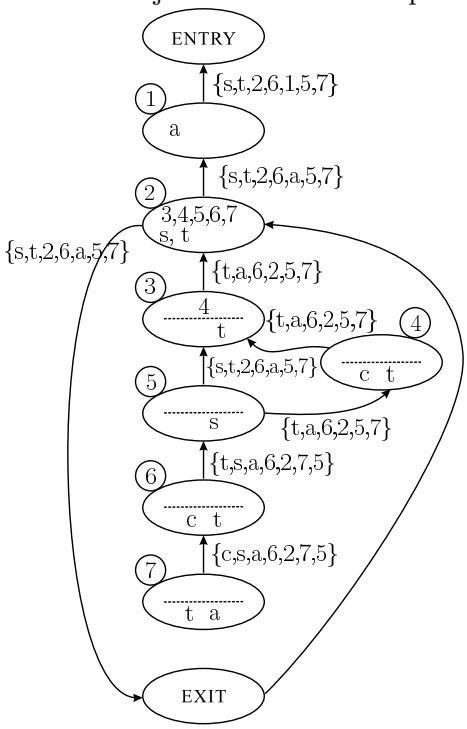

Figure 7. The final state of the process network.

1. a=0	1. a=0
2. while(s<t) {	2. while(s<t) {
3. if (t==4)	3.
4. c=t;	4.
5. s=2;	5. s=2;
6. c=t+7;	6. c=t+7;
7. t=a+4;	7. t=a+4;
}	}
The Original program	The Slice on ({c},7)

Figure 8. The original Program and Slice.

4. Discussion

The parallel slicing algorithm may be adopted to construct slices for multiple slicing criteria. The algorithm also produces some extra information about the slices it constructs, as a by product.

4.1. Simultaneous slicing

For the slicing criterion (V, n), process communication is initiated by outputting V as a message on all output channels from process n. In some situations it may be desirable to construct a slice which preserves the effect of the original program simultaneously on several slicing criteria. In such a situation, the slice would be constructed with respect to a set of m slicing criteria, $\{(V_1, n_1), \ldots, (V_m, n_m)\}$. In order to use parallel slicing algorithm to construct a slice with respect to $\{(V_1, n_1), \ldots, (V_m, n_m)\}$, process communication is initiated simply by outputting V1 at n1, V2 at n2 and so on up to V_m at n_m. This can also be achieved using the PDG approach [11] but has not previously been considered using Weiser's approach [25, 27].

4.2. Equivalence classes of slicing criteria

The final state of the network also has an interesting property concerning the sets of variable names labelling each arc of the stable process network.

Consider slicing on the criterion (V, n). For each arc a, in network, consider the restriction of each labelling to just variable names (i.e. ignore the node identifiers). Now

these sets represent the variables whose values must be presented at arc a, in order to p preserve the values of V, a, n [17]. The slice constructed by the parallel slicing algorithm, for the slicing criterion (V, n) is thuds also a slice for all (K, m), where K is outputs $(V, n)(m)$ the union of all message output by node m when slicing using the parallel slicing algorithm with respect to (V, n). Thus constructing a single slice at one point in the program, automatically produces an entire set of slicing criteria for which the program produced is also a slice.

For any program therefore, an equivalence relation $\sim$ on slicing criteria can be defined, such that $(V, n) \sim (K, m)$ if and only if they both lead to the same slice. The parallel slicing algorithm has the property that the set of labelling of stable process networks are all in the same equivalence class with respect to $\sim$.

5. Conclusion

In this paper a new algorithm for static slicing has been introduced. The algorithm exploits the natural parallelism in the control flow graph and is essentially a parallel version of Weisers algorithm based on the view of a process network as a set of recursion equations.

The authors believe that the view of a CFG as a process network with its correspondence to a set of recursion equations is extremely attractive and may form the basis for parallel solutions to other graph theoretic problems in source code analysis.

Acknowledgements

The author thanks anonymous referees for several helpful comments, suggestions and questions. The author is also thankful to Prof. H.C. Pande, Vice-Chancellor Emeritus, Prof S. K. Mukhrerjee, Vice-Chancellor and Prof. N. C. Mahnati, Professor & Head Dept. of Applied Mathematics, Birla Institute of Technology, Mesra for encouragement and support.

References

[1] Abramsky, S., Reasoning about Concurrent Systems in Distributed Computing, University of London, pp.307-319, 1984.

[2] Agrawal, H. and Horgan, J. R., *Dynamic program slicing*, in Proc. ACM SIGPLAN Conf. On Programming Language Design and Implementation, New York, 246-256, 1990.

[3] Aho, A.V., Sethi, R. and Ullman, J. D., Compilers: Principles, Technique and Tools, Addision Weseley, Reading MA, 1986.

[4] Bieman, J. M. and Ott, L. M., *Measuring functional cohesion*, IEEE Trans. Software Engineering, Vol.20, pp.644-657, 1994.

[5] Gallagher, K. B. and Lyle, J. R., *Using program slicing based on dependence graphs*, Proc. IEEE Conf. On Software Engineering, Vol.17, pp.191-200, 1991.

[6] Gopal, R., *Dynamic program slicing based on dependence graphs*, Proc. IEEE Conf. on Software Maintenance, pp.181-190, 1991.

[7] Hecht, M. S., Flow Analysis of Computer Program, Elsevier, Amsterdam, 1977.

[8] Hoare, C. A. R., Communicating Sequential Processes, Prentice-Hall, Englewood Cliffs, NJ, 1985.

[9] Horwitz, S., Prins, J. and Reps, T., *Integrating non — interfering versions of programs*, ACM Trans. Programming Languages Systems, Vol.11, pp.345-387, 1989.

[10] Horwitz, S. and Reps, T., *The use of program dependence graphs in software Engineering*, Proc. 14th Internat. Conf. On Software Engineering, Melbourne, Australia, pp.392-411, 1992.

[11] Horwitz, S., Reps, T. and Binkley, *Inter procedural slicing using dependence graphs*, Proc. ACM SIGPLAN Conf. on Programming Language Design and Implementation Atlanta, GA (1988), pp.25-46; also in: SIGPLAN Notices Vol.23, pp.35-46, 1988.

[12] Jiang, J., Zhou, X. and Robson, D. J., *Program slicing for C — The problems in implementation*, Proc. IEEE Conf. on Software Maintenance, pp.182-190, 1991.

[13] Kamkar, M., Shanmehri, N. and Fritzson, P., *Inter procedural dynamic slicing*, Proc. 4th Conf on Programming Language Implementation and Logic Programming, pp.380-384, 1992.

[14] Korel, B. and Laski, J., *Dynamic program slicing*, Inform. Process. Lett., Vol.29, pp.155-163, 1988.

[15] Leung, H. K. N. and Reghbati, H. K., *Comments on program slicing*, IEEE Trans Software Engineering, Vol.13, pp.1370-1371, 1987.

[16] Lyle, J. R. and Weiser, M., *Automatic program bug location by program slicing*, Proc. 2nd Internat. Conf. on Computers and Applications, pp.877-882, 1987.

[17] Mishra, P. K., *A parallel algorithm for static program slicing*, 72nd Annual Session of the National Academy of Sciences, India, NEHU Shilong, 25th-27th October, 2002.

[18] Mishra, P. K., *Using program slicing to simplify testing*, J. of Parallel Processing, Vol.43, 175-181, 2001.

[19] Mishra, P. K. and Sharma, C. K., Cohesion metrics in 6th International Conference on High Performance Computing, The Leela Palace Bangalore (India) Dec 16th-19th , 2002.

[20] Ott, L.M. and Thuss, J. J., *Slice based metrics for estimating cohesion*, Proc. IEEE-CS Internat. Metrics Symp., pp.78-81, 1993.

[21] Ottenstein K. J. and Ottenstein, L. M., *The program dependence graph in software development environments*, SIGPLAN Notices, Vol.19, pp.177-184, 1984.

[22] Qi, L., Fubo, Z. and Jianua, Q., *Program slicing*, J. Comput. Sci. Technology, Vol.3, pp.29-39, 1988.

[23] Tip, F., Generation of program analysis tools, Ph.D. Thesis. Centrum Voor Wiskunde en Informatica, Amsterdam, 1995.

[24] Venkatesh, G. A., *The semantic approach to program slicing*, Proc. ACM SIGPLAN Conf. On Programming Language Design and Implementation, Toronto, Canada, pp.26-28, 1991. also in :SIGPLAN Notices, Vol.26, pp.107-119, 1991.

[25] Weiser, M., Program slices: Formal, psychological and practical investigations of an automatic program abstraction method, Ph. D. Thesis, University of Michigan, Ann Arbor, MI, 1979.

[26] Weiser, M., *Programmers use slicing when debugging*, Comm. ACM, Vol.25, pp.446-452, 1982.

[27] Weiser, M., *Program slicing*, IEEE trans. Software Engineering, Vol.10, pp.352-357, 1984.

Author's Information

Dr. P. K. Mishra is Assistant Professor in the Dept. of Applied Mathematics, Birla Institute of Technology, Mesra-Ranchi(India). Dr. Mishra did his Ph.D. in Parallel Computing from APS University (India). Dr. Mishra is a Associate Editor of IJED (An International Journal of IIT Roorkee(India)) and a Senior Member of IEEE, IEEE Computer Society and IEEE Communication Society (USA).

Dept. of Applied Mathematics, Birla Institute of Technology, Mesra, Ranchi-835215, India.

E-mail: pkmishra@bitmesra.ac.in Tel: +91-651-2276406