

Detecting Concurrently Executed Pairs of Statements Using an Adapted MHP Algorithm*

Zhenqiang Chen
Department of Computer Science
and Engineering,
Southeast University,
Nanjing 210096
China

Baowen Xu
Department of Computer Science
and Engineering,
Southeast University,
Nanjing 210096
China
bwxu@seu.edu.cn

Huiming Yu
Department of Computer Science,
North Carolina A&T State University,
Greensboro, NC 27411
USA
cshmyu@ncat.edu

1. ABSTRACT

Information about which pairs of statements in a program can be executed concurrently is important for improving the accuracy of dataflow analysis, optimizing programs, and detecting errors. This paper presents a new method named Concurrent Control Flow Graph for representing concurrent Ada 95 programs in a simple and precise way. Based on this method, we develop an Adapted MHP algorithm that can statically detect all pairs of statements that may be executed concurrently. This algorithm checks not only whether a rendezvous can be triggered, but also whether it can be finished. Although this algorithm generates a conservative superset of the perfect pairs of statements, it is more precise than many existing methods.

1.1 Keywords

Concurrent program, control flow graph, MHP algorithm, program analysis

2. INTRODUCTION

Ada 95 is an object-oriented programming language that supports the construction of long-lived, highly reliable software systems [7,9,15]. The execution of an Ada program may consist of one or more tasks. Each task represents a separate thread of control that proceeds independently. Because the behaviors of concurrent Ada 95 programs are unpredictable, testing, understanding, and debugging these

kinds of programs is difficult. Thus, methods to analyze the possible behaviors of those programs are needed.

In general, it is a NP problem to perfectly detect all pairs of statements that may be executed concurrently [13]. Thus, most researches in this field are to find a feasible algorithm, which can obtain more precise information in an acceptable time, i.e. improve the efficiency by depressing the precision [1,4,6,10-12]. Masticola and Ryder proposed a non-concurrency algorithm that computes a conservative set of pairs of communication statements that never happen in parallel in a concurrent Ada program, and the complement of this set is a conservative approximation of the set of pairs that may occur in parallel [10]. Naumovich and Avruninwe proposed a MHP algorithm for detecting the statement pairs that may be executed concurrently, and has shown that the MHP algorithm can obtain more precise information than any other methods. The worst-case time cost is $O(n^6)$ [11].

To obtain more precise information in less time, we develop a new representation method named Concurrent Control Flow Graph for concurrent Ada programs, and an Adapted MHP algorithm to detect current executable statements. This algorithm yields more precise information than any methods we know, and the worst-case time cost is $O(n^5)$.

The rest of this paper is organized as follows. Section 2 introduces the general program representation method. The Concurrent Control Flow Graph representation method is presented in section 3. The concurrent features are displayed by the Concurrent Control Flow Graph. Section 4 presents preliminary notions used in the Adapted MHP algorithm. The Adapted MHP algorithm is discussed in section 5. Experimental results are given in section 6. The applications of the information obtained from the Adapted MHP algorithm is given in section 7. Conclusion remarks are given in section 8.

3. CONTROL FLOW GRAPHS

A Control flow graph is a common way to represent a sequential program [2,3,5,8]. For a program P, the control flow graph is a direct graph $CFG = \langle S_S, S_E, s_i, s_f \rangle$, where S_S

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page.

To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGAda 2001 09/01 Bloomington, MN, USA
© 2001 ACM 1-58113-392-8/01/0009...\$5.00

* This work was supported in part by the National Science Foundation of China (NSFC) (60073012).

are statements or predicate expressions. S_E is an edge set. For two nodes s_1 and s_2 , if s_2 might be executed just after the execution of s_1 , edge $\langle s_1, s_2 \rangle$ belongs to S_E . Two special nodes s_I and s_F are distinguished; s_I and s_F represent the entry and the exit of the program respectively. If $\langle s_1, s_2 \rangle \in S_E$, s_1 is a direct predecessor of s_2 and s_2 is a direct successor of s_1 , denoted by $\text{Pre}(s_1, s_2)$. All successors of a node s are denoted by $\text{Succ}(s)$.

In the CFG of a sequential program, if statement s_2 may be immediately executed after s_1 , edge $\langle s_1, s_2 \rangle$ will belong to S_E . If applying this definition to concurrent programs, the CFG will be too complex to analyze and understand, because the executions of statements are unpredicted, and it is difficult to decide whether one statement may be executed after another. At the same time, the control flows among concurrently executed tasks are not independent, because of inter-task synchronization and communications. It is not enough to represent a concurrent program only by a CFG. Thus we introduce *concurrent control flow graph* to represent concurrent tasks.

4. PROGRAM REPRESENTATION

Ada 95 provides a complete facility for supporting concurrent programming that includes tasks, entry, protected objects, select, delay, requeue and abort statements [7]. To analyze a concurrent Ada 95 program, all these facilities must be represented in a simple and precise way without losing useful information. We developed a method named Concurrent Control Flow Graph (CCFG) to represent concurrent Ada 95 programs. A CCFG of a program consists of a collection modified Control Flow Graph. Each of them represents a single task.

The main feature of the CCFG is that the original edges from the entry calls to its direct successors are removed. It can help to obtain more precise information when analyzing concurrent programs.

4.1 Representing Concurrency

In the CCFG, when a task object is activated, its body starts to be executed. The CFG of the task body is connected to the master's CFG by concurrency edges according to the rules of the activation and termination of a task. Figure 1 shows a connection of concurrent tasks of the program segment. In this figure, nodes *cobegin* and *coend* are

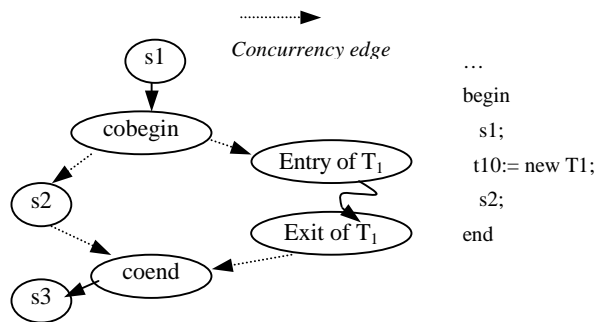


Figure 1 CCFG of Concurrency

introduced to represent the begin and end of the concurrency.

4.2 Representing abort, delay and requeue Statements

In the CCFG an *abort* statement is represented as a simple statement. The delayed time of a *delay* statement is not considered, because we can not determine how long it might sleep in static analysis. A *requeue* statement is treated as an entry call statement, and its direct successor is the exit node of the CFG of the inner most program unit that includes the statement.

4.3 Representing Rendezvous

In Ada, tasks can interact in two ways that are directly and indirectly. Using rendezvous tasks can send messages to each other directly. By shared data tasks can communicate indirectly. Rendezvous is a kind of synchronization in nature. It occurs when a task calls another task's entry. An entry family is treated as a single entry, because the index is not determined in static analysis. A client task that calls an entry of a server task may be blocked and placed in a queue. When a server task accepts the entry call from a client task, the two tasks are in rendezvous. Between the beginning and the end of the rendezvous, data may be exchanged via parameters. When the rendezvous is over, the two tasks continue their execution in parallel.

For each rendezvous, two nodes are added that are rendezvous begin node and rendezvous end node to represent the begin and the end of the rendezvous. Rendezvous nodes are connected to the tasks' CFGs by *synchronization edges*. Figure 2 shows the connection of one rendezvous. The edge from the entry call statement to its direct successor is removed from the CFG of the task, and an edge from the entry node to the start node of the rendezvous is added to represent the rendezvous. If more than one task calls the entry of a task which includes more than one accept statements, synchronization edges are added from each entry call statement to all possible rendezvous begin nodes. Because we can not assume the execution orders and which entry call can join a rendezvous of an accept statement.

4.4 Representing Select Statements

In Ada 95 [7], select statements are used for uncertainty

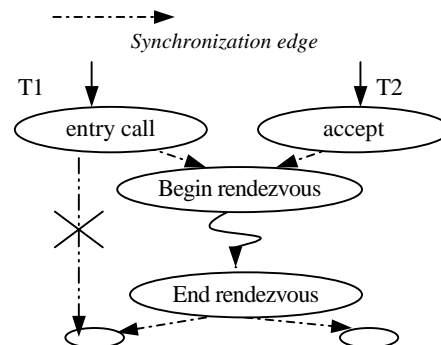


Figure 2 CCFG of a rendezvous

message passing. There are four forms of select statements. *Select accept* provides a selective wait for one or more select alternatives, the selection may depend on conditions associated with each alternative of select accept. A *Timed entry call* issues an entry call that is cancelled if the call (or a requeue-with-abort of the call) is not selected before the expiration time is reached. *Conditional entry call* issues an entry call that is then cancelled if it is not selected immediately (or if a requeue-with-abort of the call is not selected immediately). An *Asynchronous Transfer of Control* provides asynchronous transfer of control upon completion of an entry call or the expiration of a delay. All kinds of select statements are represented as statements with branches.

5. PRELIMINARY NOTIONS

Synchronization between tasks is the key inner factor that influences the execution sequence. The different schedule strategies of operation systems are not considered in this paper. We assume that each active task can be scheduled at any time.

According to the semantics of rendezvous [7], the two tasks attached to the rendezvous will wait until the rendezvous is finished or aborted by other tasks. To determine whether a rendezvous can be finished or not two sets are introduced that are PE and PT. PE represents a necessary rendezvous node set, and PT represents a necessary task set.

In a CCFG, let e_b be a rendezvous begin node and e_e be the corresponding rendezvous end node. If there is a rendezvous begin node e_{kb} (excluding e_b) on every path from e_b to e_e , e_{kb} is a *necessary rendezvous node* of e_e . All necessary rendezvous nodes of e_e consist of the necessary rendezvous node set, denoted by $PE(e_e)$.

If a rendezvous begin node has only two direct predecessors, the two tasks that include the two predecessors are the necessary tasks, otherwise, only the task that includes the *accept* statement is a necessary task. The union of all necessary tasks of the rendezvous begin nodes on a path in a rendezvous is the necessary tasks of the path. The intersection of the necessary tasks of all paths from a rendezvous begin node e_b to the end node e_e is the *necessary tasks* of e_e , denoted by $PT(e_e)$.

A task can not begin another rendezvous before finishing the current rendezvous. Therefore, from the definition of PE and PT, two rules are defined as follows:

Rule1: Let e_e be a rendezvous end node, if there is a task attached to e_e in $PT(e_e)$, the two tasks will wait forever, i.e. this rendezvous can never be finished normally.

Rule2: Let e_e be a rendezvous end node, if there is a rendezvous in the $PE(e_e)$, which can not be triggered or finished, the rendezvous represented by s can never be finished normally.

In a CCFG, a rendezvous begin node might be connected by more than one entry call node. To distinguish which

entry caller can join the rendezvous, we set a flag, Trigger, for each edge from the entry call node to the rendezvous begin node. The TRUE value of a Trigger means that the two tasks connected by the edge can join the rendezvous. Initially, all Trigger flags are set FALSE.

Intuitively, a rendezvous represented by a rendezvous begin node can take place only if both tasks attached to the rendezvous are ready to participate in it.

For two statements s_1 and s_2 , we introduce a predicate $Con(s_1, s_2)$. $Con(s_1, s_2)$ is TRUE iff s_1 and s_2 may be executed concurrently. Obviously, $Con(s_1, s_2)$ is symmetric and intransitive.

6. THE ADAPTED MHP ALGORITHM

MHP is an efficient algorithm that can detect statement pairs that may happen in parallel. It is first proposed by Naumovich and et al [11] and they show the algorithm can obtain more precise information than any other previous methods, and its worst-case time bound is $O(n^6)$. In this paper, based on the CCFG we develop an Adapted MHP algorithm to detect concurrent executed pairs of statements in Ada 95 programs. This algorithm obtains more precise information than any other methods that we know, and its worst-case time bound is $O(n^5)$.

6.1 Algorithm

The Adapted MHP algorithm associates three node sets that are GEN(s), IN(s) and M(s) with each node s in the CCFG. For all the three sets are computed repeatedly, M(s) is used to record the current approximation to the set of nodes that may be executed concurrently with s . It is the union of GEN(s) and IN(s). GEN(s) represents the nodes we can place in M(s) based on the information of s , such as whether a rendezvous can be triggered or finished. IN(s) represents the nodes that can be added to M(s) using information of the predecessors of s , such as the two tasks begin to be executed concurrently after the rendezvous.

Initially, all three sets for all nodes are empty. These sets are repeatedly computed until they do not change. At this point set M(s) represents a conservative overestimate of nodes with which node s may be executed concurrently. The sets GEN and IN are computed in each iteration of the algorithm.

According to the semantic of activating tasks, in the CCFG, all tasks connected by concurrent edges with the same cobegin node may be executed concurrently. According to the semantic of the task rendezvous, the entry caller and the acceptor continues execution in concurrency after the rendezvous. The MHP algorithm checks only whether a rendezvous can take place. However if the rendezvous can be finished the two tasks are blocked, the successor statements can not be executed forever, thus they can not be executed concurrently with any other statement. To get more precise information, in our algorithm we check not only whether a rendezvous can be triggered, but also

whether it can be finished. The procedure to compute the GEN set of a node is shown in Algorithm. 1.

```

procedure ComputeGEN(Node s) is
begin
  if s is a rendezvous node, then
    GEN(s) =  $\Phi$ ;
  else
    Let  $s_{init}$  be a node, which connects to other nodes by concurrency edges;
    if s is a direct successor of  $s_{init}$ , then
      GEN(s) = Succ( $s_{init}$ ) - {s};
    end if;
    if s is a direct successor of a rendezvous end node  $e_{end}$  then
      if the rendezvous ending with  $e_{end}$  can not be finished then
        GEN(s) =  $\Phi$ ;
      else
        Let t be the acceptor task of the rendezvous ending with  $e_{end}$ ;
        Let  $e_{begin}$  be the corresponding begin node of  $e_{end}$ ;
        if s is not a statement of task t, then
          Let ts be the task including s;
          if ts can trigger this rendezvous, and ts does not belong to PT( $e_{end}$ ), then
            GEN(s) = {successors of  $e_{end}$  in t};
          else
            GEN(s) = { $s_m$  where  $s_m$  are successors of  $e_{end}$ , and the task including  $s_m$ 
              can trigger  $e_{end}$ } - {successors of  $e_{end}$  in task included in PT( $e_{end}$ )};
          end if;
        end if;
      end if;
    end if;
  end if;
end ComputeGEN;

```

Algorithm 1 Procedure ComputeGEN

```

procedure ComputeIN(Node s) is
begin
  if s is a rendezvous begin node, then
    Let  $s_a$  be the corresponding accept statement,  $M_{temp}$  be a node set initialized as null;
    for each edge  $\langle s_e, s \rangle$  connected to s do
      if its Trigger is TRUE then
         $M_{temp} = \cup M(s_e)$ ;
      end if;
    end for;
    IN(s) = M(s)  $\cap$   $M_{temp}$ 
  else
    IN (s) =  $\cup M(s_p)$  where  $s_p$  are the direct predecessors of s;
  end if;
  if the direct predecessor  $s_p$  of s is an abort statement then
    for each task t aborted by  $s_p$  do
      Remove the nodes of t from IN(s);
    end for;
  end if;
end ComputeIN;

```

Algorithm 2 Procedure ComputerIN

In Ada, tasks can be executed at varying rates. If a node s_p can be executed concurrently with another node s_m , for the successor s of s_p , if s is not a rendezvous node, s may be executed just after the execution of s_p . Thus, s can be executed concurrently with s_m . If s is a rendezvous node, only when the rendezvous can be triggered, i.e. the accept statement and the corresponding entry call node can be executed concurrently, s may be executed concurrently with s_m . The IN set of a node is computed by procedure ComputeIN shown in Algorithm 2.

In every iteration of the Adapted MHP algorithm, we set $M(s) = IN(s) \cup GEN(s)$. For Con (s_1, s_2) is symmetric, set $M(s_1) = M(s_1) \cup \{s_2\}$ when $s_1 \in M(s_2)$. For a node s , the IN sets of its successors may be changed with the change of $M(s)$. Thus, its successors are added to the worklist, and

```

procedure AdaptedMHP(Node  $s_i$ ) is
    ----  $s_i$  is the entry node of the CCFG.
begin
    Let the set M for all nodes be empty;
    Compute PE and PT for each rendezvous end node;
    Let all Trigger flags be FALSE;
     $W = \{s_{i1}, s_{i2}, \dots, s_{ik}\}$ ,
        where  $s_{ij}$  are the nodes connected to  $s_i$ ;
    repeat
        Remove a node  $s$  from  $W$ ; Let  $M_{old} := M(s)$ ;
        if  $s$  is a rendezvous begin node then
            Let  $s_a$  be the accept statement,
            Let  $s_{p1}, s_{p2}, \dots, s_{pm}$  be predecessors of  $s$ 
            except  $s_a$ ;
            for each  $s_{pi}$  do
                if  $s_{pi} \in M(s_a) \vee s_a \in M(s_{pi})$  then
                    Set the Trigger flag of edge  $\langle s_{pi}, s \rangle$ 
                    to be TRUE;
                end if;
            end for;
        end if;
        ComputeGEN( $s$ );
        ComputeIN( $s$ );
         $M(s) = GEN(s) \cup IN(s)$ ;
        if  $M_{old} \neq M(s)$  then
            for each  $s_m \in M(s) - M_{old}$  do
                 $M(s_m) = M(s_m) \cup \{s\}$ 
                 $W = W \cup Succ(s_m)$ 
            end for;
             $W = W \cup Succ(s)$ 
        end if;
    until  $W = \Phi$ ;
    for each node  $s_i$  do
         $MHP(s_i) = M(s_i)$ ;
    end for;
end MHP;

```

Algorithm 3 Procedure AdaptedMHP

computed for another iteration. The Adapted MHP algorithm is shown in Algorithm 3.

6.2 Time Cost of the Adapted MHP Algorithm

In this section we analyze the time cost of the Adapted MHP algorithm. Applied the Adapted MHP algorithm was applied to an Ada program with n statements. According to the definition of the CCFG, the number of nodes in a CCFG is linear with n . A statement may be executed concurrently with all n statements, i.e. the nodes in the three sets M, IN and GEN for each node are finite. Since the three sets increase monotonically, the Adapted MHP algorithm will eventually terminate. We can prove that the worst-case time bound of this algorithm is $O(n^5)$.

In the CCFG, each node has n predecessors at most. The union or intersection of the two node-sets can be computed in $O(n)$. Thus, the worst-case time cost of procedure ComputeGEN and ComputeIN is $O(n^2)$, and every iteration of the procedure AdaptedMHP can be finished in $O(n^2)$. In the procedure AdaptedMHP, a node may enter another iteration if one of its predecessors' M set is changed. There are $O(n)$ nodes at most in a node's M set and M increases monotonically. Thus, intuitively, a node may be computed at most $O(n^2)$ times. The total time cost is $O(n) * O(n^2) * (n^2) = O(n^5)$.

6.3 Adapted MHP for Subprogram Calls

In this section we discuss how the Adapted MHP algorithm handles subprogram calls. Let P be a subprogram that does not include entry call statements. If s is a call node for subprogram P , any node in the body of P may be executed concurrently with any node in $MHP(s)$. A subprogram may be called concurrently by more than one task, the executions of multiple instances of this method may overlap in time. In this case, the MHP set of a statement may include itself.

In the case of subprograms containing entry call statements, the bodies of the subprograms are embedded in the caller, and analyzed with the caller together.

According to the semantic of protected operations, only protected functions can be executed concurrently with other functions of the same protected object.

7. EXPERIMENTAL RESULTS

We have implemented the Adapted MHP algorithm in the "Ada95 Reverse Engineering and Software Maintenance Supporting System (ARMS)". It works as part of the slicing and program-optimizing tools. We have applied it to 54 Ada programs. The experimental results show that in most cases the time cost is linear with the line number of the source code. We also find that with more synchronization among tasks, there is less efficiency and accuracy.

Consider the program in Figure 3. Three tasks are declared. T2 has an entry Entry1. T1 and T3 are synchronized with

```

procedure Main is
  task T1;
  task T3;
  task T2 is
    entry Entry1;
  end;
  task body T1 is
    begin
      s0;
      T2.Entry1;
      s1;
      s2;
    end T1;

  task body T2 is
    begin
      s3;
      accept Entry1;
      s4;
    end T2;

  task body T3 is
    begin
      s5;
      T2.Entry1;
      s6;
    end T3;
  begin s7; end;

```

Figure 3 An Ada95 program

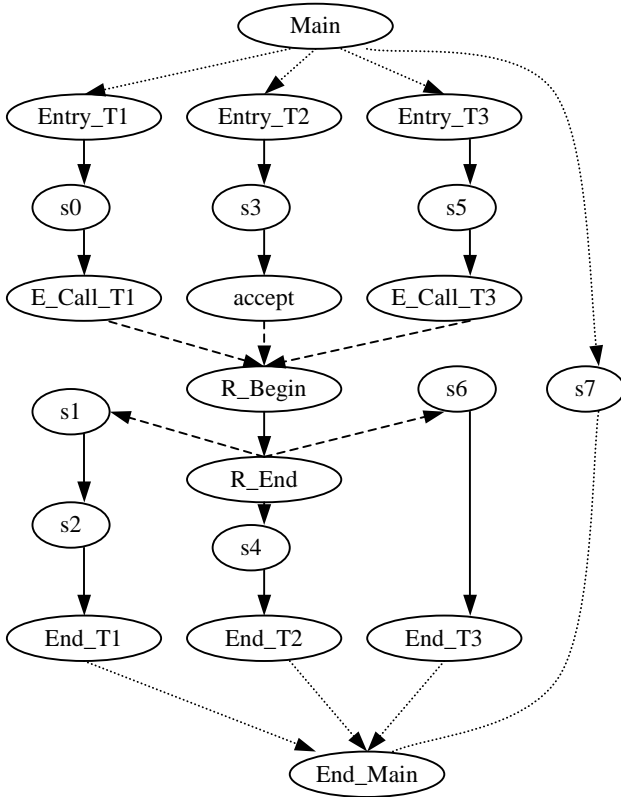


Figure 4 The CCFG of the program

T2 by calling its Entry1. Figure 4 shows the CCFG of this program. All the three tasks are activated at the start statement of Main procedure. Thus, the concurrent control flow graphs are connected to Main procedure by concurrent edges. For the rendezvous, two nodes R_Begin and R_End are added to represent the begin and end of the rendezvous. The three rendezvous nodes E_Call_T1, accept and E_Call_T3 connect to R_Begin by synchronization edges. Only after the synchronization, control flows return to the tasks. Thus, we remove edges $\langle E_Call_T1, s1 \rangle$ and $\langle E_Call_T3, s6 \rangle$ from the CFGs of T1 and T3.

According to the ComputeGEN procedure, the entry nodes of the tasks can be executed concurrently, i.e. $GEN(Entry_T1) = \{Entry_T2, Entry_T3, s7\}$, $GEN(Entry_T2) = \{Entry_T1, Entry_T3, s7\}$, $GEN(Entry_T3) = \{Entry_T1, Entry_T2, s7\}$.

According to procedure ComputeIN, $IN(Entry_T1) = IN(Entry_T2) = IN(Entry_T3) = \Phi$. For node s0, $IN(s0) = \{Entry_T2, Entry_T3, s7\}$, $GEN(s0) = \Phi$, $MHP(s0) = IN(s0) \cup GEN(s0) = \{Entry_T2, Entry_T3, s7\}$.

After several iterations of using the Adapted MHP algorithm the result shows that E_Call_T1 (E_Call_T2) can be executed concurrently with the accept statement, and the two rendezvous can be triggered and finished. Thus, $GEN(s1) = \{s4\}$, $GEN(s6) = \{s4\}$. Because $s1 \notin MHP(R_End)$, $s6 \notin MHP(R_End)$, $s6 \notin IN(s1)$, $s1 \notin IN(s6)$, s1 and s6 can not be executed concurrently.

8. APPLICATIONS

The CCFG and the Adapted MHP algorithm have been applied to analyze concurrent programs. In this section, we will show how to use the CCFG and the Adapted MHP algorithm to detect dead statements, errors, and improve the accuracy of dependence analysis.

8.1 Detecting Dead Statements

In concurrent programs, some statements may never be executed because of the schedule and synchronization. Such statements are called dead statements in this paper. Dead statements have no influence on the program except make the program more difficult to understand.

8.2 Detecting Errors

Obviously, if a node s can not be reached from the entry node of the CCFG, s can never be executed. If the rendezvous can not be triggered or finished, the two tasks will be blocked. If s is the rendezvous begin node, the statements that take s as a predominate node would never be executed.

In concurrent programs, the concurrent accession of shared data should be under control. If all shared data are accessed by synchronization, the efficiency of the whole program will be depressed. It is difficult to check whether a statement is synchronized. Using the Adapted MHP algorithm, we can obtain all pairs of statements that may be executed in concurrency. Analyzing such pairs, it is easy to find all statements that need to be protected. If two tasks never attempt to access a shared item at the same time, any unnecessary locking operations can be removed.

8.2 Detecting Errors

From the Adapted MHP algorithm, when the control flow of a task reaches an entry call or accept statement with Trigger that is FALSE, or a rendezvous which can not be finished normally, the task will be blocked forever until it is aborted. It is much easier to find the reasons of these errors using the information obtained from the Adapted MHP algorithm.

8.3 Improving the Accuracy of Dependence

Analysis

Dependence analysis is an important method to analyze, debug, test and maintain programs [2,3,5,8,14]. In the analysis of sequential programs, the data dependence can be computed by analysis along the control flow graph. However in concurrent programs, the non-deterministic execution of statements may lead to too many paths to analyze or lead to imprecise information. Using the Adapted MHP algorithm we can obtain more precise data dependence information by the following three steps: 1) Remove dead statement nodes and related edges from the CCFG. 2) Compute data dependencies using the CCFG refined by step1. 3) If $\text{Con}(s_m, s_n)$ holds, s_m and s_n may be executed concurrently with each other, s_n may be executed just after the execution of s_m in some schedules. Thus, if s_n refers variables defined at s_m , s_n data depends on s_m .

9. CONCLUSIONS

With more and more concurrent programs being used in practice, information of which pairs of statements can be executed concurrently becomes very important. The information can be widely used in optimization, detection of anomalies, and improving the accuracy of data flow analysis.

In this paper, we present a new method named Concurrent Control Flow Graph to represent concurrent Ada programs according to the semantic of Ada 95 programming language. It describes the concurrent facilities of Ada 95 in a simple and precise way without losing useful information. Based on this method, we develop an Adapted MHP algorithm to detect all pairs of statement that can be executed concurrently. This algorithm checks not only whether a rendezvous can be triggered, but also whether it can be finished. Thus, this algorithm can generate more precise information than the best MHP algorithm that we know. The worst-case time bound of the Adapted MHP algorithm is $O(n^5)$. The experimental results show that this algorithm is efficient and precise.

10. REFERENCES

[1] Callahan, D., and Subhlok, J. Static analysis of low level synchronization. In Proceedings of the SIGPLAN/SIGOPS Workshop on Parallel and Distributed Debugging, 100-111, 1988.

- [2] Chen, Z. , and Xu, B. Dependence Analysis and Static Slices of Concurrent Programs. ICYCS'99, Nanjing, China, 239-242,1999
- [3] Chen, Z., Xu, B. , and et al. An Approach to Analysing Dependency of Concurrent Programs. APAQS'2000, IEEE Press, Hongkong, 34-39, 2000.
- [4] Chen, Z. , and Xu, B. Detecting All Pairs of Statements in Parallel Programs, NCYCS'2000, Nanjing, China, 265-269.
- [5] Chen, Z. , and Xu, B. Slicing Concurrent Java Programs. ACM SIGPLAN Notice, 2001, 36(4): 41-47.
- [6] Duesterwald, E. and et al. Concurrency analysis in the presence of procedures using a data flow framework. In Proceedings of the ACM SIGSOFT Fourth Workshop on Software Testing, Analysis, and Verification, pages 36-48, Victoria, B.C., October 1991.
- [7] ISO/IEC 8652:1995(E). Ada Reference Manual- Language and Standard Libraries.1995.
- [8] Krinke, J. Static Slicing of Threaded Programs. ACM SIGPLAN Notices, 33(7): 35-42, 1998.
- [9] Li , B., Xu, B., and Yu, H. Transforming Ada Serving Tasks into Protected Objects. SIGAda'98, Washington, 240-245, 1998.
- [10]Masticola, S., and Ryder, B. Non-concurrency analysis. In Proceedings of the Twelfth of Symposium on Principles and Practices of Parallel Programming, San Diego, CA, May 1993.
- [11]Naumovich, G., and Avrunin, .S. A Conservative Data Flow Algorithm for Detecting all Pairs of Statements that may Happen in Parallel, ACM Press, Proceedings of the 6th International Symposium on the Foundations of Software Engineering, 24-34, November 1998.
- [12]Naumovich, G., and et al. An Efficient Algorithm for computing MHP Information for Concurrent Java Programs. ESEC/FSE '99. pp. 338-354, September 1999.
- [13]Taylor, R.N. Complexity of analyzing the synchronization structure of concurrent programs. Acta Informatica, 19:57-84, 1983.
- [14]Xu, B. Reverse Program flow Dependency Analysis and Applications. Chinese J. Computers, 16(5): 385-392, 1993.
- [15]Xu, B. An Overview of Ada95. Journal of Computer Research and Development, 34(1): 53-57, 1997.

