PERIODICA POLYTECHNICA SER. EL. ENG. VOL. 39, NO. 1, PP. 27-36 (1995)

CONTROL FLOW CHECKING IN MULTITASKING SYSTEMS

István MAJZIK and András PATARICZA

Department of Measurement and Instrument Engineering
Technical University of Budapest
H-1521 Budapest, Műegyetem rkp. 9, Hungary
Phone: +36-1-463-2057
E-mail: majzik@mmt.bme.hu

Received: November, 1994

Abstract

The control flow checking technique presented in our paper is based on the new watchdog-processor method SEIS[1] (Signature Encoded Instruction Stream). This method is intended to check the still uncovered area of state-of-the-art microprocessors using on-chip caches or instruction pipelines, since the processor instruction bus needs not be monitored. The control flow is checked using assigned actual signatures and embedded reference signatures. Since the actual and reference signatures are embedded in the checked program, the usual reference database and the time-consuming search/compare engine in the watchdog can be omitted. The evaluation of the actual signature is a simple combinatorial task allowing high speed and thus the sharing of the watchdog between different tasks and processors. The checking method has been extended to higher levels of the application like simultaneous check of different processes and their synchronization in multitasking systems.

Keywords: fault tolerant computing, concurrent error detection, control flow checking, watchdog processors.

1. Introduction

Multiprocessing systems that run computing intensive applications require a high level of fault tolerance, thus the early error detection is a key design factor. The majority of failures is caused by transient faults. Experiences and fault injection based simulations had shown that up to 70 % of the transient faults result in the disturbance of the program control flow (CZECK and SIEWIOREK, 1990). One of the widely used methods for the concurrent checking of the program control flow is the application of *watchdog processors* (WP (MAHMOOD and MCCLUSKEY, 1988)).

[1]This research is part of the Hungarian-German Joint Scientific Research Project #70 with additional support from: SFB 182 (DFG), OTKA-760, T-4494 and F7414 (Hungarian NSF).

A watchdog processor is a relatively simple coprocessor which monitors the control flow of the checked processor using its state parameters. The run-time processor state and the reference control flow are represented by *signatures*. Two types of watchdog processors are distinguished on the basis of the generation of the run-time signatures: WPs working with *assigned* or *derived signatures*.

The *derived signatures* (e.g. in WILKEN and SHEN, 1988) are generated directly by the WP which monitors the address and instruction bus of the checked processor and compacts the values of the signals as binary vectors (e.g. using linear feedback shift registers). The *reference signatures* are computed or compacted before the program run in the same way.

In order to generate the derived signatures, the processor instruction bus needs to be monitored. In the case of up-to-date microprocessors using built-in instruction cache and prefetch queue this approach cannot be used. The only way to identify the processor state is to *assign signatures* to some states of the program. The high level program text is preprocessed before the compilation and signature transfer instructions are inserted. In run-time, these instructions transfer the signatures to the WP. The checked processor and the WP are running simultaneously, the WP compares the run-time signatures with the reference ones downloaded before the program start.

The until now developed assigned signature based WP methods (in the following referred to as AS methods) use either a simple *watchdog reference program* (LU, 1982) or a *reference database* (MICHEL and HOHL, 1991). The watchdog program contains the control structures of the checked program with the modification that the computations are replaced by signature receive and check instructions. The reference database is the adjacency matrix of the control flow graph of the checked program stored in some sparse matrix format. The disadvantage of both approaches is that due to the complicated evaluation of the run-time signatures (execution of the watchdog program or search in a large-scale database) the hardware implementation of the WP needs an independent microprocessor. On the other hand, the usefulness of these methods in multitasking systems is limited, because the size of the reference becomes large if the number of running processes increases.

One of the cooperation projects of the Department of Measurement and Instrument Engineering (Technical University of Budapest) and the Department of Computer Structures (University of Erlangen) is the development of a high-speed watchdog processor. The goals and guidelines of the project were as follows:

- Development of a WP working with assigned signatures (according to the type of the checked system) and being able to check multitasking systems as well;
- Reducing the size of the reference database to an acceptable level (even if the number of running processes is high);
- Assuring a fast signature evaluation which is executable by a simple sequential circuit (without the need of an independent microprocessor).

In the following first the basics of the signature assignment are discussed, then the SEIS algorithm for computing of the signatures is outlined (Section 3). The checking of the procedure calls and the synchronization of cooperating processes are presented in the next two sections. The support of error recovery and the hardware architecture of the WP are covered by Section 6 and Section 7. At the end, measurement results and conclusions are presented.

2. Signature Assignment Basics

The control flow of a (high-level) program is the execution sequence of the statements. The set of all possible fault-free executions of a program is associated with a *control-flow graph* (CFG), where the vertices represent the statements (or given statement sequences) of the program, the edges represent the admissible control operations between them, independently of the data dependencies. In procedural programming languages each procedure can be associated with a control-flow graph.

According to the basic AS scheme, signatures are assigned to the vertices of the CFG, which is often referred to as the *encoding of the CFG*. The WP monitors the signature flow by checking whether an edge exists between the vertices associated with the actual and the previously received signatures. At a higher level, the change between the CFGs of different processes and processors can be examined, if the CFG identification information is embedded in the run-time signatures as well. In this way the procedure calling mechanism, the scheduling, the synchronization of cooperating processes can be checked in the WP.

The main goal of our new algorithm called *Signature Encoded Instruction Stream* (SEIS (PATARICZA et al, 1993)) is the reduction or elimination of the large-scale reference database and the complicated WP program used in the former methods. The run-time signatures should contain all the information which is needed in order to evaluate them. In SEIS, an assigned signature not only identifies the current state of the program but

also contains information of the possible successor states (i.e. signatures). In this way each signature can be validated on the basis of its predecessor in the signature flow, the possible successors of each signature need not be stored explicitly in the WP. The state of the checking in a given CFG is represented by a single reference signature which is regularly replaced by the actually evaluated and fault-free one. The switching between the checked CFGs is simple, since only the actual reference signature has to be saved and replaced with the reference of the new CFG (returning to the checking of the actual CFG, the saved reference is used again).

Since each signature should contain information of the identification of the possible successor ones, the limited signature size in real computer systems and the desired high signature transfer rate need the limitation of the number of successors. It is (more or less) possible since in control structures of commonly used programming languages the number of successors of a statement is limited by the syntax (in most cases it is 2 or less; in assembly languages only simple two-way branches and loops can be used). However, there are control structures in which the number of successors (or predecessors) of a given statement is not limited (e.g. the *case* branches in a *switch* structure). To encode the CFG of such 'irregular' control structures, the encoding algorithm has to be carefully defined. The SEIS preprocessor (for programs in C language) assures that each signature has the same size, independently of the number of successors to be identified.

3. Encoding and Checking of the Control Flow Graph

The input of the SEIS preprocessor is the original high-level program source, the output is the modified program text which contains the inserted statements which transfer the run-time signatures to the WP. The preprocessed program can be compiled by the original compiler of the language and it can be executed without further modifications.

The steps of the SEIS encoding algorithm are informally presented as follows ((MAJZIK, 1994) is a more formal description):

1. The control flow graph of the program is extracted. The basic control structures are assigned *elementary control subgraphs* which can be composed hierarchically. The subgraphs are defined in such a way that they satisfy the requirements of the encoding: the number of successors of the vertices should be limited.
2. The edges of the CFG are collected into an edge list (since the base of the encoding and checking is the existence of edges in the graph). The listing of the edges can be reduced to the well-known problem of the *Eulerian circuit generation.* Inserting additional edges, the CFG

can be transformed into an Euler graph in which the Eulerian circuit can be generated using simple linear algorithms. This circuit contains all edges once and only once.

3. The vertices listed in the Eulerian circuit are encoded. A cyclic ordering of the possible code values is defined. Subsequent vertices in the circuit which are connected by normal edges of the CFG are assigned subsequent codes; if they are connected by additional edges then they are separated skipping a code value. In the Eulerian circuit, each vertex is listed several times (the number of occurrences of a vertex is equal to the maximum number of its input and output edges). The signature of a vertex is generated concatenating the code values assigned to the multiple occurrences of it (called in the following *sublabels*). This encoding method defines the *checking rule* as well: A signature is a valid successor of a reference one if and only if one of its sublabels is successor (in the used cyclic ordering of the sublabel codes) of one of the sublabels of the reference signature.
4. In some vertices, the number of sublabels is reduced. The vertices of irregular control structures, which can have an unlimited number of successors (predecessors), are marked as *multiple output* (*multiple input*, respectively) vertices in the control-flow graph extraction step. (These types of vertices, their predecessors or successors can be identified on the basis of the syntax of the language.) The number of sublabels of the multiple input (multiple output) vertices have to be reduced. The base of the reduction is that the sublabels referring to the same successor (predecessor) vertex may have the same code if they have no predecessor (successor) sublabels. In this way the output (input) edges of the multiple output (input) vertices can be connected to the same sublabel reducing the number of sublabels. These special vertices are defined in such a way that the reduction is always possible.

The SEIS graph extraction and encoding algorithm assures that if programs in C language are preprocessed then the number of sublabels of each vertex is limited to 3. The edges which are encoded using less than 3 sublabels are completed appending one of the existing sublabels to the signature once again. The fact that each signature consists of a fixed number of sublabels enables a simple implementation of the WP hardware. The base of the signature evaluation is the rule described in Step 3.

As an example let's consider the following procedure:

```
int test_procedure() {
    for(f=0; f<9; f++) {
       if (a<b) a=b+f;
       else  c=a-f;
       }
    }
```

The preprocessed version of this procedure is as follows:

```
int test_procedure() {
    SEND_SIG(SOP,1,10,1); {
    for(f=0; f<9; f++) {
       SEND_SIG(NRM,2,5,2); {
       if (a<b) {SEND_SIG(NRM,3,3,3); a=b+f;}
       else {SEND_SIG(NRM,6,6,6); c=a-f;}
       SEND_SIG(NRM,4,7,4); }
    }}
    SEND_SIG(EOP,8,11,8);
}
```

The SEND_SIG macro transfers the signature to the watchdog, its implementation depends on the hardware interface between the checked processor and the watchdog. The parameters of the macro are the signature type (see later) and the sublabel codes. The signatures belonging to adjacent statements contain subsequent sublabel codes, in this example the successor function is the one which increases the code by one.

4. Checking of the Procedure Calls

There are two ways to check the procedure calls. First, *intermediate signatures* can be inserted before and after the procedure calls connecting the first and last signature of the called procedure to the reference of the caller environment. In this way the caller and the called procedures have a single common CFG.

The second method allows the use of independent CFGs of the called procedures. In the WP, similarly to the checked program, a *signature stack* is implemented which stores the reference signatures of the embedded procedure calls. The initial and final vertices of the procedures are marked by *Start of Procedure* (SOP) and *End of Procedure* (EOP) flags. Receiving a SOP signature the watchdog stores the actual reference signature on the stack (signature push), the first reference of the new CFG is the actual

and unchecked SOP signature. In the case of an EOP signature the most recently saved reference is restored from the stack (signature pop operation) and checking of the CFG of the caller procedure resumes.

5. Checking the Synchronization of Processes

The synchronization of processes is performed by the scheduler and by the synchronous communication. If a *process identifier* is appended to each signature then the scheduler can easily be checked. Changing the running process the scheduler transfers the ID of the actual process to the watchdog. The WP stores it internally and compares with the identifiers embedded in the run-time signatures. Only the signatures of the actual running process are valid.

The communication can be checked by using *guard signatures*. They are transferred to the WP in the same way as the normal signatures, but their effect and evaluation are different. Two types of guards are used: *start guard*, inserted before a communication statement, and *checker guard*, inserted after it. The processes beginning a synchronous communication initialise their *communication registers* in the WP by the start guard signatures. Receiving a checker guard signature, it is evaluated in the WP fault-free if all the processes which are partners in the communication have already updated their communication registers by sending the initialisation guard. (I.e. a process is enabled to continue to run if its partners have already begun the communication, too.) If only two-way communications are enabled then the structure of the guard signatures and their evaluation are simple.

6. Support of Error Recovery

If an error of a checked process is detected then the system is interrupted and a status word is available in the WP which contains the detailed description of the error. The system can restart the execution of the erroneous process using a previously saved state of it, which is called *checkpoint* state (rollback recovery).

The prototype of the SEIS WP supports the rollback recovery of the checked system. When the main processor stores a checkpoint then the state of the actual process is saved simultaneously internally in the WP (initiated by a special command similar to the guard signatures). The WP-internal state of a process is represented by its signature stack, so it has to be stored as checkpoint. If a rollback recovery is performed then the checkpoint stack space replaces the actual one, in this way the execution

as well as the checking of the process can continue from the fault-free previous state.

7. The Checker Hardware

The signatures assigned to the processes of a multitasking application consist of 5 parts: the 3 sublabels, the procedure and the process identification numbers (the use of procedure ID is optional; in multiprocessor systems processor ID can be used as well). These parts are evaluated by autonomous modules of the WP hardware:

1. The *statement level module* checks the actually received signature on the basis of the reference one. The possible sublabel pairs are examined by 9 comparators and a combinatorial sublabel successor function. The time needed to evaluate a signature is the delay of the comparators. After the evaluation, the reference signature is updated.
2. The *procedure level module* checks the switching between the CFGs using the signature stack. The SOP and EOP signatures initiate a push or pop operation updating the stack pointer and the reference signature.
3. In the *process level module*, the process ID is compared with the reference ID of the running process which was transferred by the scheduler. The guard signatures initialize and check the communication registers. The signature transfer is monitored by a timer which detects when a process fails to send signatures.

The WP was built using standard programmable logic devices (MACH series of AMD). The procedure stack was implemented in a 256K RAM which was shared dynamically between the processes (this stack proved to be oversized if no recursive procedure calls were checked). The WP can check multiple processors in a time-sharing manner as a coprocessor on the 32 bit VME bus. The transfer and evaluation of a signature takes about 500 ns.

8. Measurement Results

The first measurements (in MAJZIK et al, 1994) have shown that there is a strong dependence between the fault coverage of this error detection method and the number of signatures sent to the WP (the time and memory overhead of the preprocessed program).

The *memory overhead* is acceptable even for programs in the MB range (in average up to 30 %). However, the *time overhead* is a critical fac-

tor. If the signature transfer needs more time than the fetch and execution of a processor instruction, then the overhead can exceed 100 %, especially in applications consisting of small iteration loops, since in each step a signature has to be transferred additionally. In order to reduce the time overhead, two types of reduction algorithms were elaborated. The first one, the *static reduction*, reduces the number of signatures eliminating some vertices in the CFG. The second one, called *dynamic reduction*, decreases the overhead caused by the overtested short loops containing the transfer of a single signature in their body; these signatures are not transferred in each execution of the loop, only at a given rate which is controlled by a dynamic reduction factor. These two reduction techniques are implemented in the preprocessor.

The fault injection experiments were executed defining different static and dynamic reduction factors. Without any reduction, the WP is able to detect up to 50–60 % of the errors not detected by the standard error detection mechanisms of the system (e.g. access to non-existent or unused memory, segmentation fault, illegal instruction). Using static reduction, the fault coverage decreases rapidly, as the time between subsequent signatures increases (especially in high-level programs where the statements can cover complex instruction sequences at the assembly level). The dynamic reduction of small factors has no such effects; while reducing the time overhead, the fault coverage remains the same as without the reduction.

9. Conclusions

On the basis of the measurements, the following applicability conditions of the SEIS method (and, in general, of the AS methods) can be derived:

- The SEIS method performs the fastest signature evaluation known among the different AS methods. The signature checker hardware needed by it is extremely simple, the checks can be extended to higher levels of the application using additional checker modules.
- The preprocessor approach assures a portable and compiler-independent signature assignment. However, just like for all AS methods, existing programs, which cannot be recompiled, cannot be checked.
- If there are standard error-detection mechanisms in the system (segmentation checks, examination of the memory access rights) then this method does not increase drastically the error detection capability.
- If the signature transfer is slow (compared with the memory access cycle in the system) then the time overhead of the preprocessed program is unacceptably high (even if the signature evaluation is fast). If the main processor uses speed-up mechanisms, like instruction prefetch

queue or cache, a slow signature transfer becomes a performance bottleneck in the checked system. (However, the SEIS and the other AS methods are intended to be used even in this type of microprocessor systems, in which the instruction bus of the processor is not observable).

References

1. CZECK, E.W. – SIEWIOREK, D.P.: Effects of Transient Gate-Level Faults on Program Behaviour. In *Proc. FTCS-20*, pp. 236–243, 1990.
2. LU, D.J.: Watchdog Processors and Structural Integrity Checking. *IEEE Trans. on Comp.*, 37(7) pp. 681–685, 1982.
3. MAJZIK, I.: SEIS: A Program Control Flow Graph Encoding Algorithm for Control Flow Checking. *Technical Report TU-TR-94-EE-14*, TU Budapest, (pp. 66), 1994.
4. MICHEL, E. – HOHL, W.: Concurrent Error Detection Using Watchdog Processors in the Multiprocessor System MEMSY. In *Fault Tolerant Computing Systems, Informatik Fachberichte 283*, pp. 54–64, Springer, Berlin, 1991.
5. MAHMOOD, A. – MCCLUSKEY, E.J.: Concurrent Error Detection Using Watchdog Processors - A Survey. *IEEE Trans. on Comp.*, 37(2) pp. 160–174, 1988.
6. MAJZIK, I. – PATARICZA, A. – DAL CIN, M. – HOHL, W. – SIEH, V.: Hierarchical Checking of Multiprocessors Using Watchdog Processors. In *Proc. EDCC-1, LNCS 852*, pp. 386–403, Springer, Berlin, 1994.
7. PATARICZA, A. – MAJZIK, I. – HOHL, W. – HOENIG, J.: Watchdog Processors in Parallel Systems. *Microprocessing and Microprogramming*, (39) pp. 69–74, 1993.
8. WILKEN, K. – SHEN, J.P.: Continuous Signature Monitoring: Efficient Concurrent Detection of Processor Control Errors. In *Proceedings of the 1988 Int. Test Conf.*, pp. 914–925, 1988.