Program Development Tools: Debugging by Reverse Computing

Kamonluk Suksen
Department of Computer Engineering
Faculty of Engineering, Chulalongkorn University,
Bangkok, Thailand
Kamonluk.Su@student.chula.ac.th

Prabhas Chongstitvatana
Department of Computer Engineering
Faculty of Engineering, Chulalongkorn University,
Bangkok, Thailand
prabhas.c@chula.ac.th

***Abstract*— More and more program development tools have turned to Web-based. It has an advantage on being a multi-platform tool. This work proposes a debugging tool based on web interface. The main idea of the tool is that the execution of the program can be regarded as a flow. This flow can be captured and played back later. Therefore, any state of the executed program can be analyzed and errors can be pinpointed. We propose a time-reversal computing as a way to capture the flow of execution so that the time step can be reversed without starting from time-zero.**

Keywords— debugging, web-based tools, reverse computing

I. INTRODUCTION

Traditionally, debugging is done via "tracing" by single stepping through the program's source code. The values of variables can be observed as the program is executing step-by-step. So, the behavior of the program can be checked to understand what went wrong. If we want to know what the states of computation are at some point in time, we have to execute the program from the beginning to that point. During debugging, this inspection of states can be laborious. To save time in repeating the whole computation each time we want to observe some particular state at some particular time, we can store all the states of execution and query them later, but the amount of storage in a practical debugging session may be prohibitively large.

Let us assume that we have a capability to perform "backward" execution of an operation, then given a current state it is easy to step-back in time by performing a "reverse" operation of the previous operation. In this case, it is possible to "go-back" to any previous time step without starting from the beginning and without storing a large amount of state of computation.

Let us imagine that during debugging of a program, P, we do have a "reverse" version of this program, called P'. By executing P' at the current state S(t), we can "reverse" the state to the previous state S(t-1). This is the use of reversible computing to go back in time.

This work proposes a design of P' in the form of a virtual machine. This virtual machine can be used in complement with tracing operation during debugging a program P and enables a debugger to go-back in time. The design of P' is dependent on the instruction set of P. Some state must be stored so that reverse execution of an instruction is possible. We show the detailed design of such machine and analyze the amount of storage necessary to run P backward. A simple example is shown to illustrate the practicality of such approach.

II. RELATED WORKS

Reversible computation has been a topic of great interest for recent years, and therefore a large amount of work has been done in widely different areas [1, 7]. However, we focused on program inversion, as programs based on a reversible computation paradigm and reversible programming language. There exist research described the Reversible Virtual Machine (RVM) that can serve as a target implementation platform for a reversible probabilistic language. The technique for making a reversible language is adding some extra state in the form of a boolean variable and a history stack [2]. An important mechanism of reversible languages is switch input and output store for inverse constructs [3].

An Omniscient debugger [5] works by recording all state changes in the run of a program. The programmer can look inside of the history of the program and go "backwards in time" to see where it was set, and what its value was. There exist bidirectional debugger [6] in which all traditional forward movement commands can be performed with equal ease in the reverse direction. It works by re-execute the program to identify and stop at the desired earlier point. A development of web based programming assistance tool for novices [4] is a tool for developing programs on web browser. It is aimed for novice programmers and it has a helpful display for errors. However, the program must starts from time-zero for reverse debugging.

III. BACKGROUND & OVERVIEW

A traditional debugger can move only in the direction of forward execution. The user specifies where the execution of the program should stop, and the debugger executes forward until it reaches that point. Forward movement is natural and

well understood, however it is often exactly the opposite of what would be most convenient for a user trying to observe the cause of an error.

If we want to know what the states of computation are at some point in time, we have to execute the program from the beginning to that point. We call this method "no-save" because we can know what the states of computation by re-execution to that point without storing information in each state. However, the re-execution wastes the time because it is running of former states that has already execute.

To save time in repeating the whole computation each time we want to track down some particular state at some particular time, we can store all the states of execution and query them later, but the amount of storage in practical debugging session may be prohibitively large. We call this method "save everything".

To reduce storage and save time in repeating the whole computation, in this paper we propose a debugging tool. The tool can capture the execution of the program and play back later by a time-reversal computing so that the time step can be reversed without starting from time-zero. Moreover, some particular state is not necessary to storage any information like the save everything method. We have name our method "reverse computing". The debugging by reverse computing offers a trade-off between storage efficiency and time efficiency. We describe the reverse computing in Section 4.

IV. Reverse Computing

The reverse computing was implemented in the virtual machine that operate on web interface. This virtual machine can be used in complement with tracing operation during debugging a program and enables a debugger to go-back in time. The design of virtual machine is dependent on the instruction of program. Some state must be stored into stack so that reverse execution of an instruction is possible.

Each state, the information that must be stored is the program counter (PC), and the instruction. The information in an instruction are: operation, address1, address2, address3. Moreover the value of registers and memory in some particular state that change after the execution must also be stored.

The operation in some state can change value in the register for example the instructions: ld, ldd, jal, ori, or, lt, pop, add, addi, sub, subi, mul, and div. The operation in some state can change value in the memory such as the instructions: st, std and stx. In addition, some arithmetic operation can write reverse arithmetic operation. Therefore, the state is that arithmetic operation does not need to store the value of register that change after execution.

A. Reverse Operation

Let us imagine that during debugging of a program P, we do have a "reverse" version of this program, called P'. By executing P' at the current state S(t), we can "reverse" the state to the previous state S(t-1) by executing reverse operation of the operation in the current state. The reverse operation will set the value of PC, operation, address1, address2, address3, register and memory to be the value of the previous state. Then it will display the output of the previous state. The amount of time to reverse computing is equal to the amount of time to execute the operation in the current state. The number of clock used in each operation are as follows.

- Operation ldd, st, std, trap, push, pop use 6 clocks
- Operation jal, addi, subi, lt, or, ori, mul, muli, div, divi use 5 clocks
- Operation jt, jf, ret use 4 clocks
- Operation ld, add, sub use 1 clock

We give a situation that a user chose to observe the output clock at 2030 in the bubble sort program and wanted to step back by previous clock, as shown in Fig.1.

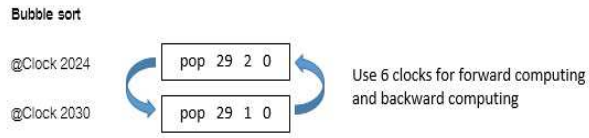

Fig. 1. The backward execution from clock at 2030 to the previous clock in bubble sort program

B. Reverse Arithmetic Operation

Some arithmetic operation can write the inverse of the operation such as add, addi, sub, subi, mul and muli. We show the reverse arithmetic operation as follows.

• add 1 2 3	R[1] = R[2] + R[3]
Reverse add 1 2 3	R[1] = R[2] – R[3]
• addi 1 2 3	R[1] = R[2] + 3
Reverse addi 1 2 3	R[1] = R[2] – 3
• sub 1 2 3	R[1] = R[2] – R[3]
Reverse sub 1 2 3	R[1] = R[2] + R[3]
• subi 1 2 3	R[1] = R[2] – 3
Reverse subi 1 2 3	R[1] = R[2] + 3
• mul 1 2 3	R[1] = R[2] * R[3]
Reverse mul 1 2 3	R[1] = R[2] / R[3]
• muli 1 2 3	R[1] = R[2] * 3
Reverse muli 1 2 3	R[1] = R[2] / 3

The reverse arithmetic operation can decrease the amount of information that needs to be stored. Because it can return the value in the register of the previous state by execute reverse of arithmetic operation in the current state without storing the previous value of the register in the previous state.

V. DEBUGGING BY REVERSE COMPUTING

A debugging session begins with running the program to a breakpoint. If the programmer wants to observe the state of previous clock, he/she can initiate the "reverse" execution. The reverse virtual machine will execute from the current instruction, results in reverting the state to the previous state. It changes the value of PC, instruction register, registers and memory to be the values of the previous state.

Diagram represented overall process of debugging by reverse computing based on web interface, as shown in Fig. 1.

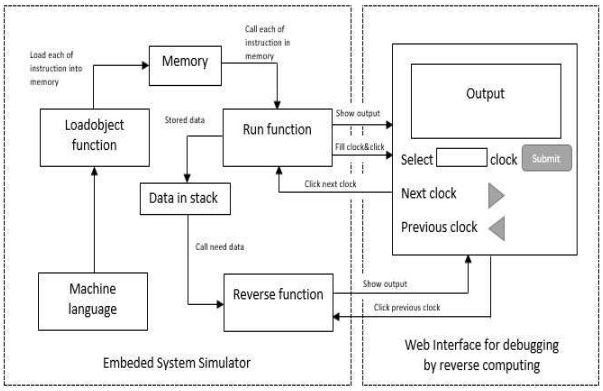

Fig. 2. Diagram represented process of debugging by reverse computing using web interface

We give an example of the operation of the debugging by reverse computing includes the following figures.

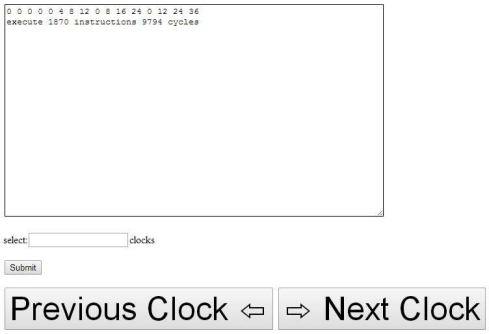

Fig. 3. The output from running the matrix multiplication program

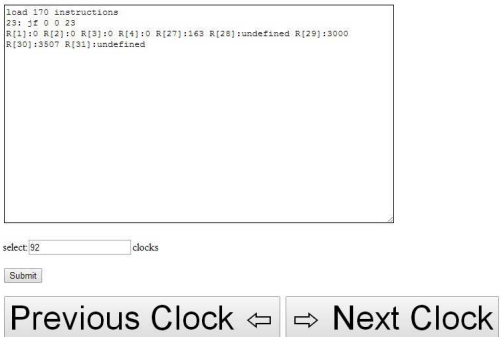

Fig. 4. The output from running the program to a breakpoint, the output clock at 92

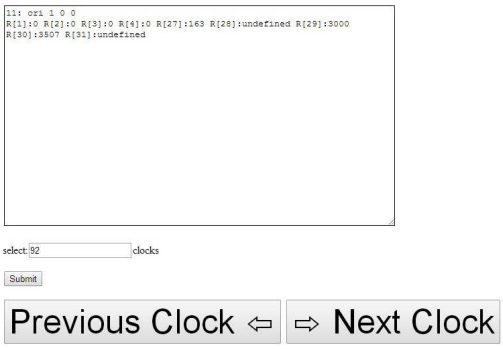

Fig. 5. The output from reverse execution that execute from the current instruction of clock at 92

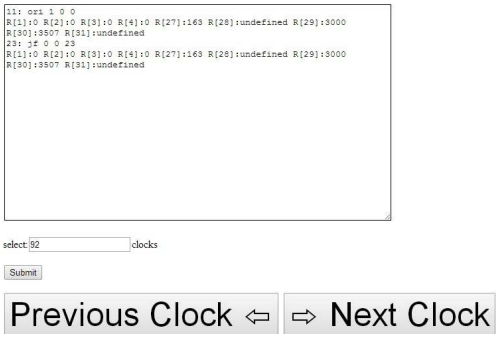

Fig. 6. The output from forward execution that continuously execute from Fig.5

VI. EXPERIMENTAL RESULT

To illustrate the effectiveness of our scheme, we compare the proposed method with two naïve implementations. First implementation is saving all states for all instructions executed. Second implementation is running from time-zero to the n-1 th instruction. In the first naïve scheme, a large amount of states will be stored. Any state from the beginning to the breakpoint can be accessed and displayed. In the second scheme, no state is stored but it requires to run the program from time-zero to n-1 th instruction.

Two simple programs are used as benchmarks: bubble sort and matrix multiplication. All three modes were tested in the situation that a user chose to observe the output clock at 2030 and wanted to step back to the previous clock. The results from the runs are summarized in Table I and Table II.

TABLE I. MEASUREMENT OF TWO RUNS: SAVE EVERYTHING AND REVERSE COMPUTING (UNIT: WORD, TIME: CLOCK)

Program	Save Everything		Reverse Computing	
	Amount of data	*Time to rerun*	*Amount of data*	*Time to rerun*
Bubble sort	63,756	0	51,643	6
Matrix multiplication	13,090	0	10,297	4

In Table I, the bubble sort program executed totally 9,108 states and the matrix multiplication 1,870 states. For the save

everything method, each state stores: PC, operation, address1, address2, address3, register and memory. Thus, the amount of information to be stored of every states in the bubble sort program and the matrix multiplication program are in total of 63,756 and 13,090 words respectively. It requires a large amount of storage to store the information in every states. However, storing all the states of execution enables users to query any state later without spending time to re-execution. For the reverse computing method, it is not necessary to store all of information in every states because each state stores only information in some particular state at some particular time. Moreover, the reverse arithmetic operation in the reverse computing method can reduce the amount of information that must be stored. Therefore, the information that needs to be stored in the reverse computing method is less than those of the save everything method.

TABLE II. MEASUREMENT OF TWO RUNS: NO-SAVE AND REVERSE COMPUTING (UNIT: WORD, TIME: CLOCK)

Program	No-Save		Reverse Computing	
	Amount of data	*Time to rerun*	*Amount of data*	*Time to rerun*
Bubble sort	0	2,024	51,643	6
Matrix multiplication	0	2,026	10,297	4

In Table II, the no-save method will not store the information of any state but it has to execute the program from the beginning to the state that users want to observe the output. Thus, the amount of information that needs to be stored of both programs is zero. However, it requires time to re-execute the program. The bubble sort program consumes time to re-execute as much as 2,024 clocks and the matrix multiplication program consumes time to re-execute 2,026 clocks. The time to re-execute is dependent on the number of clock that users want to observe. When executing to a very distant clock, seeing the output of the previous clock will definitely take long time to re-execute. Moreover, re-execution wastes the time because it is a executing of a previous state that has already been executed. The reverse computing method will run the reverse operation of the operation in the current state to go-back to any previous time step without starting from the beginning. Therefore, the amount of time to re-execute is equal to the amount of time to execute the operation in the current state, as shown in Fig.1 and Fig.7.

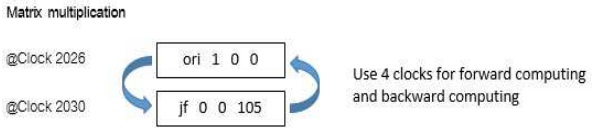

Fig. 7. The backward execution from clock at 2030 to the previous clock in matrix multiplication program

From both programs, the proposed method can reduce the amount of information that will be stored by 20.17% compared to the save-everything method. It also reduces the time by 99.75% compared to the run-from-beginning method.

VII. CONCLUSION

We have developed a reverse operation of data and arithmetic operation and demonstrated through a working implementation that it is possible to build a debugging tool by reverse computing. The method of reverse computing allows a debugger to locate and move back to any states without starting from the beginning and without storing a large amount of state data.

The efficiency of our scheme is illustrated by comparing it with two naïve implementations. First implementation is the method that save all states of all instructions executed. Second implementation is the method that starts its execution from time-zero to the n-1 th instruction. In the first scheme, a large amount of states will be stored. Any state from the beginning to the breakpoint can be accessed and displayed. In the second scheme, no states is stored but it requires to run the program from time-zero to n-1 th instruction. Two simple programs are used as benchmarks: bubble sort and matrix multiplication. All of these three modes were tested in the situation that users chose to track down the output and then step back to the previous clock. The proposed method can reduce the amount of information that will be stored by 20.17% compared to the save-everything method. It also reduces the time by 99.75% compared to the run-from-beginning method.

The proposed debugging tool can be used as a traditional forward movement debugging. It also can be performed with equal ease in the reverse direction. Moreover, the tool is based on web interface so it would be most convenient way to perform debugging on any platform that supports web interface.

REFERENCES

[1] Axelsen H.B., Gluck R. “What do reversible programs compute?” (2011) Lecture Notes in Computer Science, 6604 LNCS, pp. 42-56.

[2] Stoddart B., Lynas R., Zeyda F., “A Virtual Machine for Supporting Reversible Probabilistic Guarded Command Languages,” (2009) Electronic Notes in Theoretical Computer Science, 253 (6), pp. 33-56.

[3] Yokoyama T., “Reversible Computation and Reversible Programming Languages,” (2009) Electronic Notes in Theoretical Computer Science, 253 (6), pp. 71-81.

[4] A. More, and J. R. Kumar, V.G., “Web Based Programming Assistance Tool for Novices,” (2011) IEEE International Conference on Technology for Education, Chennai, Tamil Nadu.

[5] Lewis B., Ducasse M. Using events to debug Java programs backwards in time (2003) Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA, pp. 96-97.

[6] Boothe B. Efficient algorithms for bidirectional debugging (2000) SIGPLAN Notices (ACM Special Interest Group on Programming Languages), 35 (5), pp. 299-310.

[7] Engblom J. A review of reverse debugging (2012) Proceedings of the Conference on System, Software, SoC and Silicon Debug, art. no. 6338149, .