Design and Implementation of SPARC V8 CPU Simulator in Virtualized Verification System

Yongchao Tao[1, a] ,Wencheng Xiang[1, b] and Xianghu Wu[1, c]

[1] Shenzhen Academy of Aerospace Technology, Shenzhen, China

[a]taoyongchao652@163.com, [b]HPEC_SAAT@163.com, [c]wxh_hit@126.com

Keywords: SPARC, all-digital simulation, instruction set simulation, interpretive execution.

Abstract. The aim of this paper is to design and develop a simulator with high efficiency and flexibility for the SPARC processor which is widely used in present domestic aerospace industry based on the virtualized verification system. Firstly, the feature of SPARC instruction set architecture is studied and the simulation of its register file, execution of instruction and handling of trap is implemented wtih C programing languge respectively. Further, the high performance general graphics processor is explored to gain more performance improvement and the pre-decoding functions are eventually realized with CUDA and obtain considerable seed up.

1 Introduction

The general instruction set simulator can be divided into interpretive and compiled types according to the different technologies. The basic explanatory instruction set simulator works on a general purpose PC and interprets and executes the machine code of the executable program of the simulated target machine[1]. Each machine code is used for fetching, decoding, and executing, combined with traps processing and other functions, and finally complete the functional simulation of the simulated processor.

SPARC is an open set of technical specifications developed by Sun Microsystems Inc[2]. The lab was invented based on David Patterson's research at the University of California at Berkeley for RISC (Reduced Instruction Set Computer) research. RISC architecture is an advanced, efficient, new type of computer architecture[3]. SPARC proposed by SUN Microsystems is one of its typical implementations. SPARC has a large number of intensive registers, the collection of the current advanced software technology, hardware technology, there are a large number of window into the register heap[4]. It can achieve multi-level instruction flow, so that the processor can almost every clock cycle on the average implementation of an instruction. SPARC processor is widely used in high-end server field and aerospace field, has great research value[5].

2 SPARC Simulation Core Design

Virtual processor components in the virtual authentication system as the core role, it determines the test target environment hardware architecture, mainly complete the test code instructions decoding and simulation of the implementation of operations, and provide the overall simulation of the basic operating environment support. According to the above requirements analysis, we can see that in order to realize SPARC simulation, on the one hand, it realizes the requirement of real SPARC processor. On the other hand, we can realize the corresponding interface according to the interface requirement of the complete virtual authentication system. According to these requirements, this paper gives the overall design as shown in Figure 1.

It can be seen from the figure, SPARC simulation core mainly need to achieve four categories of functional modules:

a) The processor performs the simulation function: The module of this part is the core part of the whole simulation kernel, which realizes the function simulation of register group simulation, instruction system simulation and trap processing simulation.

b) Processor-related debugging functions: GDB as a good open source debugger provides us with a powerful debugging function. In this paper, the upper part of the virtual authentication system from the GDB improved, and therefore inherited its powerful debugging capabilities. However, many debugging functions are closely related to the processor, so the need for processor simulation kernel to achieve the relevant functions.

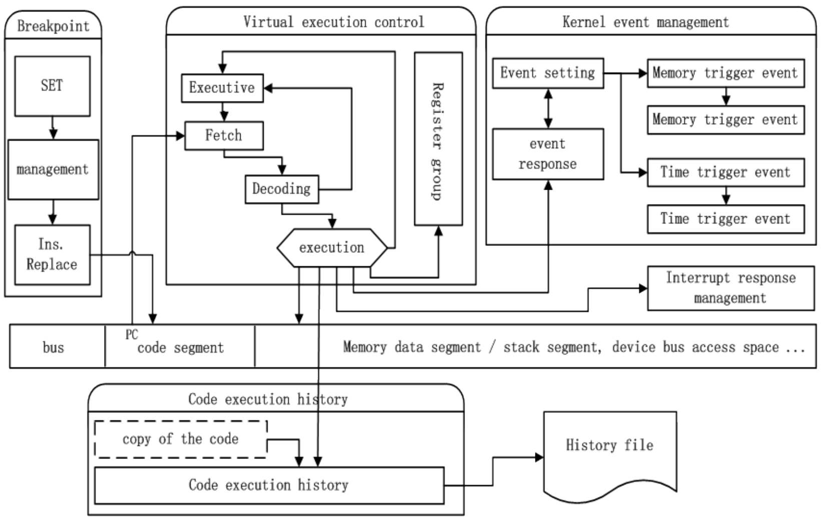

Fig.1. Processor Simulation Core Overall Design

c) Processor related auxiliary functions: In addition to the debugging capabilities inherited from GDB, the virtual authentication system in this article also provides coverage analysis, as well as kernel time management for fault injection and data acquisition, which also requires the processor emulation core to provide the underlying functionality support.

d) Device access function: During the execution of the program, in addition to the need to access the processor internal registers, but also need to access external devices, including memory, timers, and serial and so on.

3 Detailed Design and Implementation

1) Instruction Execution Design and Implementation

Decoding operations usually need to decode the option-code, operand, and through the operation code to find the instruction to simulate the implementation of the function of the process. According to the previous analysis, the use of interpretive techniques in this paper, usually using a main loop structure, in the circulation of the body constantly fetching, translation refers to, and the implementation. Take the SPARC V8 instruction as an example, remove a machine instruction from memory, parse the option-code and operand of the instruction, and then use the option-code to jump to the simulation execution function of the instruction. The simulation execution function will handle the State changes, and perform related calculations. As shown in Figure 2.

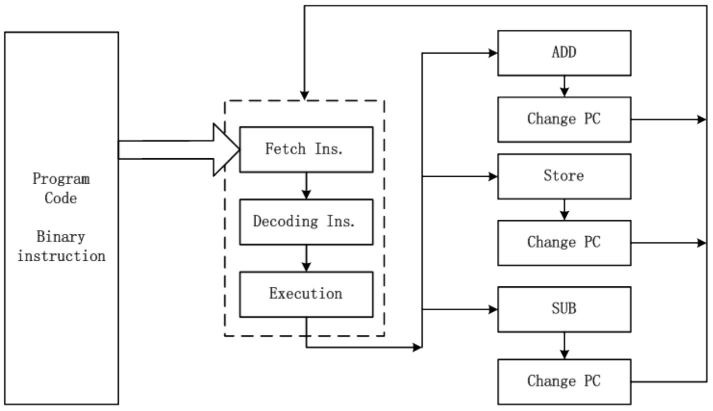

Fig.2. Decode execution control

2) Instruction Decoding and Disassembly Design and Implementation

General serial preprocessing: For the ordinary serial pre-decoding algorithm, the implementation of the process is a parse of machine instructions, analytical methods and instructions in the previous instruction in the same resolution.

CUDA parallel pre-decoding: CUDA is a parallel programming model and software environment; developers can use a powerful GPU for general graphics (GGPU). Compared with the ordinary serial algorithm, the parallel algorithm fully explores the independence of the instructions in the pre-decoding stage, and greatly improves the efficiency of the pre-decoding by using the GPU parallel parser code.

3) Design and Implementation of Trap Processing

Trap processing is divided into two operations, respectively, for the trap submission and trap response. If an exception occurs during instruction execution or an interruption occurs while querying an external device interrupt, the trap information needs to be submitted to the processor emulation core via the trap commit operation, and update the relevant data in the simulation kernel data structure, in order to prepare for the trap response. When an instruction is executed and all external device interrupt information has been queried, the processor emulates the trap information submitted before it is processed, If there is a hardware action to change the relevant processor state, and jump to the trap processing entry to continue to run. Figure 3 for the trap to deal with specific flow chart.

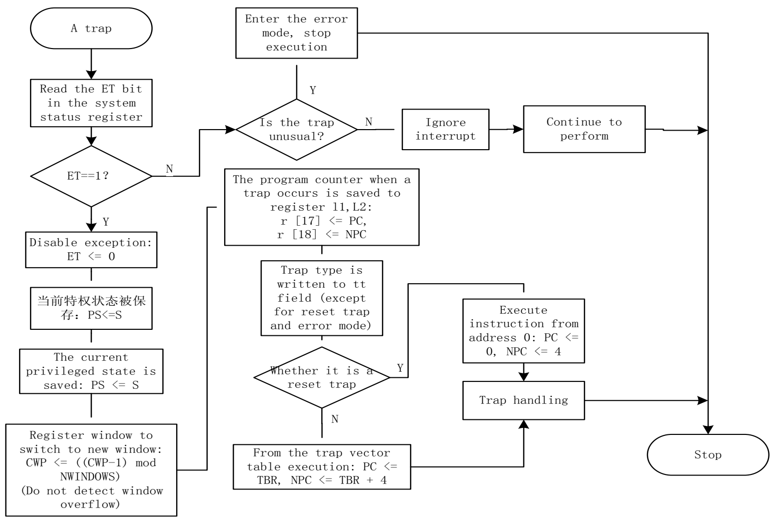
A trap
Read the ET bit in the system status register
ET==1?
Y
N
Disable exception: ET <= 0
当前特权状态被保存：PS<=S
The current privileged state is saved: PS <= S
Register window to switch to new window: CWP <= ((CWP-1) mod NWINDOWS) (Do not detect window overflow)
Enter the error mode, stop execution
Y
Is the trap unusual?
N
Ignore interrupt
Continue to perform
The program counter when a trap occurs is saved to register l1,L2: r [17] <= PC, r [18] <= NPC
Trap type is written to tt field (except for reset trap and error mode)
Whether it is a reset trap
Y
N
From the trap vector table execution: PC <= TBR, NPC <= TBR + 4
Execute instruction from address 0: PC <= 0, NPC <= 4
Trap handling
Stop

Fig.3. Trap processing flow

4) Realization of Peterson Algorithm

The Peterson algorithm is a concurrency algorithm that implements software for two or more tasks without conflicting mutual access to single resources, it uses shared memory for communication, improved by Gray L. Peterson in 1981.

The algorithm uses two variables, flag and turn. Where flag [n] is equal to true indicates that the nth task wishes to enter the critical region. Only when the task P1 does not require access to the critical area or P1 by turning the turn set to 0 and give priority to P0, the task P0 can enter the critical area. But it is still challenging to implement a Peterson algorithm on a modern multiprocessor system with a loose storage model, such as x86. In these systems, in order to improve operational efficiency, the storage write operation can be out of order, the return value of the read operation may be speculative, If you do not special handling, then in the case of concurrent, the Peterson algorithm may not be able to normal implementation.

In addition to the processor reordering the instructions, but also to prevent the compiler to optimize the code caused by the need to flag and turn variables are declared as volatile, told the compiler these two variables may be modified by other tasks at any time. Each time they use them, they must be read from memory again, rather than reading the temporary value of the register, and the resulting algorithm is shown as fellow.

Initialization
volatile bool flag[2] = {false, false};
volatile int turn;	

TASK 0：	TASK 1：

Into the critical area
P0:   flag[0] = true;
P0_gate: turn = 1;
mfence//Memory
  while (flag[1] &&   turn == 1);
	Into the critical area
	P1: flag[1] = true;
	P1_gate: turn = 0;
	    mfence//Memory barrier
	    while (flag[0] && turn == 0);

// Task 0 Critical Area Code	// Task 1 Critical Area Code
Exit the critical area flag[0] = false;	Exit the critical area flag[1] = false;

4 SPARC Simulation Core Test and Performance Analysis

For the SPARC simulation kernel function test, the most important thing is to ensure that the simulation of the instructions can be correctly resolved execution, this paper a total of 161 SPARC V8 instructions. But because in practice, most of the instructions are rarely used, this test case, the test covers the commonly used 81 instructions, the specific instructions and the number of instructions for each instruction as shown in Table 1.

Table 1.Instructions and Executions

Ins.	executions	Ins.	executions	Ins.	executions
nop	1271258004	sra	3069802	restore	6743645
ba	1171943432	bl	3056956	std	5398944
ld	245964287	bne	2590546	ldd	5398153
or	135524363	ldub	2417178	call	4144174
sethi	128208941	srl	1825282	bleu	3477615
add	104294209	rdpsr	1505937	lduh	135308
subcc	85013829	sub	1367409	rdtbr	94057
st	60189976	rdwim	1275926	andn	88324

Processor emulation has become an important part of today's processor design and software development, and its efficiency has also become a key factor affecting the performance of the entire system. In order to improve the speed of simulation, a large number of recent research focused on the use of compiled simulation technology, but the compiler has a number of restrictions, in many applications can't be used, In this paper, the use of interpretive simulation technology, but the basic interpretation of simulation technology is generally low efficiency, this article has carried out a number of optimization, to obtain a higher simulation speed.

The performance of the serial implementation and CUDA implementation is as follows:

Table 2.Serial/CUDA program run time(ms)

Numbers	Serial running time	CUDA running time
1000	703	1759
2000	1280	2191
4000	2586	2493
8000	5184	3619
10000	6145	3945
15000	9998	5375
15751	10572	5194

As can be seen from the table above, For the comparison between the serial algorithm and the running time of the CUDA algorithm, it can be seen that when the number of instructions of the pre-decoding is small, the CUDA algorithm needs to run a large amount of data in the host and device segments before running, resulting in a longer start-up time , The decoding efficiency is lower than the serial algorithm; However, when the number of pre-decoding instructions is gradually increased, the drawbacks of the serial algorithm are improved, and the processing time increases rapidly as the number of instructions increases. In contrast, the CUDA algorithm increases the processing time slowly and gradually shows its parallel acceleration the advantages.

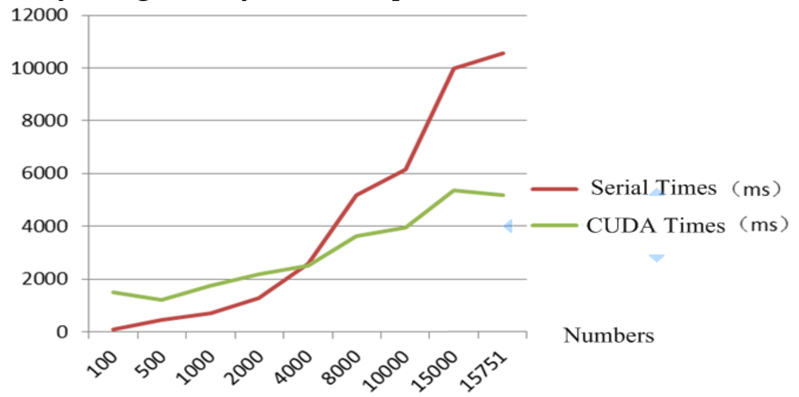

Fig.4. Comparison between Serial Algorithm and CUDA Algorithm Program Run Time

5 Conclusion

This paper designs and implements a simulator of SPARC V8 processor which is widely used in China's aerospace field for general embedded virtual verification system. The main results are as follows: the implementation of the implementation of the technology, achieve the SPARC V8 instruction set commonly used in the 161 command function simulation, and trap processing, equipment management to achieve a simulation, with high flexibility. In order to further improve the speed of simulation execution, this paper uses the pre-decoding technology to explore and propose to use CUDA general graphics computing technology to its parallel acceleration, and achieved good acceleration effect.

Acknowledgement

In this paper, the research was sponsored by the future industry special funds of Shenzhen (Project No. CXZZ20140718154349249).

References

[1] Turing A M. On Computable Numbers, with an Application to the Entscheidungsproblem[J]. Proceedings of the London Mathematical Society, 1936, 42(1):230-265.

[2] Gill S. The Diagnosis of Mistakes in Programmes on the EDSAC[J]. Proceedings of the Royal Society A Mathematical Physical & Engineering Sciences, 1951, 206(1087):538-554.

[3] Asad K, Weiqiang M, Chris W, et al. Multi-Threaded Simics SystemC Virtual Platform[C], ICCAD '15 Proceedings of the IEEE/ACM International Conference on Computer-Aided Design. Piscataway : IEEE, 2015: 373-379

[4] Achim N, Gunnar B, Oliver S, et al. A universal technique for fast and flexible instruction-set architecture simulation[C], Design Automation Conference. New Orleans : IEEE, 2002:22-27.

[5] Pidgeon A, Dawe G, Dartnell A. Software emulators: a virtual processor to support training simulations[C]. Proceedings of DASIA. Dublin : ESA, 2002:13-16.