Dynamic Binary Translator for Full Virtualizing Spacecraft Computer

Hyeona Jeong[†1] Hyunwoo Joe[†1] Cheolsoon Kwon[†1] Hyungshin Kim[†1,*]

Software reusability for spacecraft platform is one of the most important requirements because of the huge re-verification cost. Re-using pre-proven software can reduce a long and cost-taking development cycle and improve the reliability as well. Full virtualization technique is fit to boost software reusability. Dynamic binary translation is a core technology to achieve full virtualization. In this paper, we developed a prototype of dynamic binary translator engine for spacecraft computer. The binary translator analyzes and translates execution files for the ERC32 processor. The ERC32 is the processor of the guest virtual machine. During translation, we try to avoid traps on the target computer since they are the main source of the virtualization overhead. The translator prototype executes generated binaries on the LEON4 processor which is the Next Generation Multicore Processor (NGMP) in space sector. In evaluation, our binary translator is about 1.3 times faster than the translator which uses trap-and-emulate method.

1. Introduction

Recently virtualization is proposed as the candidate technology for safety-critical space application. As functional requirements are increased, functional units which comprise the subsystems are distributed physically and it makes the platform big and complex[1]. Aeronautic field suffered similar problems and introduced Integrated Modular Avionics (IMA)[2] Since European Space Agency (ESA) moved this concept to space, virtualization has become one solution for IMA. XtratuM[3] and PikeOS[4] which can support ARINC 653 partitioning for IMA has been reported.

Spacecraft computer has its own characteristics. When the spacecraft is visible from the ground, the ground station can communicate the spacecraft within limited time. Thus, the spacecraft should be operated autonomously in orbit. These features require that the flight software must be completely verified in order to guarantee mission success before it is launched. Whenever a new spacecraft is developed, the inherited software, which has been proven for previous missions, should be redeveloped or modified. Hence, software reusability without modification is one of the most important considerations in space sector because of the huge re-verification cost. In other words, complete software reusability can reduce the development cost. Full virtualization technique provides the excellent compatibility[5], but most of spacecraft virtualization use para-virtualization technique due to the poor performance of full virtualization even though it has poor compatibility.

According to Popek and Goldberg[6], traditionally critical instructions can be performed by trap-and-emulate method for full virtualization. But VMware claimed that trap is very expensive operation. They have tried to handle sensitive instructions using binary translation and improved the performance by reducing number of traps[7]. Moreover, traps may make real-time tasks non-deterministic. Recently, hardware assisted virtualization technique has been introduced in desktop, server and even mobile embedded system such as Intel VT-x[10] and Cortex A15[11]. However, in space sector, there is no hardware extension for virtualization. In [8][9], they showed VMEXIT has large overhead and [8] used binary analysis and translation technique for reducing VMEXITs.

In this paper, we describe implementation of DBT for proof-of-concept with the full virtualizing spacecraft hypervisor prototype. In addition, we discuss why we adopted full virtualization technique for spacecraft computer and how we will overcome the overhead by implementing the binary translator in the future.

The DBT analyzes and translates execution files for the ERC32 processor. The ERC32 is the processor of the guest virtual machine. During translation we try to avoid traps on the target computer since they are the main source of the virtualization overhead. The prototype executes translated binaries on the LEON4 processor[13] which is the ESA Next Generation Multicore Processor (NGMP). In evaluation, our binary translator executed faster than the traditional trap-and-emulate one. Configuration of Page Layout

2. Related Works

Virtualization technologies are widely used for desktop and server markets. VMware successfully virtualized x86 processors with DBT earlier. However, DBT has overhead during translation. Xen[14] and XtratuM have addressed the performance using paravirtualization instead of full virtualization with DBT. But they require modifying and recompiling guest operating system and applications. KVM[15], Xen, VMware, and VirtualBox[16] use the hardware assist for virtualization. But there is no hardware extension for virtualization in space sector.

Hypervisors in aerospace support IMA such as ARINC 653[17]. XtratuM and PikeOS are known results. They are bare-metal hypervisor using paravirtualization. For space use, they support RTEMS[18] and processors such as LEON2 and LEON3, which are widely used on spacecraft computers. However, they use paravirtualization to achieve low virtualization overhead instead of reusability. Flight software (FSW) from previous missions has to be modified and recompiled for their implementation. In other words, if a new spacecraft has IMA and full virtualization architecture, the flight

†1 Dept. of Computer Science and Engineering at Chungnam National University.
{ jha113, jhwzero, cskwon }@cnu.ac.kr
* corresponding author : hyungshin@cnu.ac.kr

software can be achieved from the pre-proven FSWs on different partition.

DBT translates binaries from one ISA to another on-the-fly. HDTrans[19] is lightweight instrumentation system and it can translate IA-32 to IA-32 based on low translation table. This technique is lightweight. However it has limited optimization chances, and cannot support multiple ISA. DynamoRIO[20] and PIN[21] translates IA-32 to IA-32. They translate binaries to IRs in first parse, and then generate binaries based on passed IRs. They can use various optimization methods for a compiler. But it could be slower because of going through parsing process.

To overcome this problem, we implemented the efficient DBT. We also designed a space hypervisor prototype using the DBT for the LEON4.

3. Hypervisor for Spacecraft

We proposed the design of spacecraft hypervisor prototype in [22]. Figure 1 shows the overall structure. We report design and prototype implementation of Type II hosted full virtualizing hypervisor for spacecraft computer. The hypervisor includes Linux kernel for various stable components. Therefore, it can be seen as a special operating system. The virtual machine monitor (VMM) is located at an operating system. A virtual machine (VM) is a user process. Each VM consists of a Virtual CPU (VCPU), virtual devices, an execution module, and the DBT.

DBT and VCPU are key features about full virtualization and this is different characteristic with paravirtualization. It is the process of translating machine code at the run time. We define a VCPU for ERC32 and classify its instructions. The VCPU is a data structure including shadow registers and windows. Execution Manager executes guest VM using the DBT and VCPU.

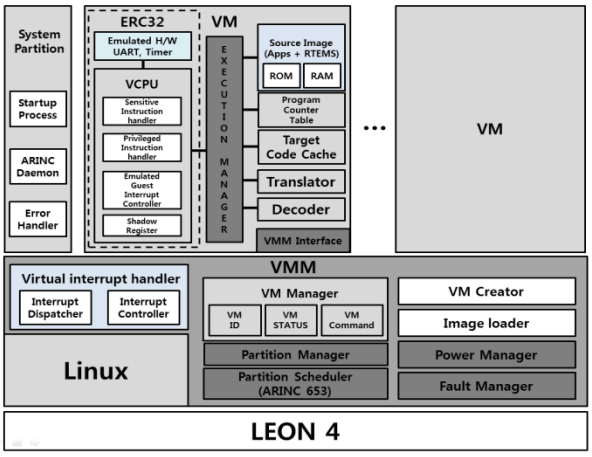

Figure 1 S-Hypervisor prototype [24]

4. Design of Dynamic Binary Translator

DBT is required to virtualize sensitive instructions in our hypervisor because classical trap-and-emulate method cannot emulate sensitive instructions which are not privileged. However, it has the disadvantage about performance due to extra step for translation. Nevertheless, some research groups like VMware achieve full virtualization with DBT for acceptable performance. They have tried to avoid traps on the host machine as many as possible by binary analysis, instrumentation, and translation[7].

DBT is still a useful tool even though the sensitive instructions have been emulated via classical trap-and-emulate approach. If hardware assist is available, it achieves better execution time. In [8], they claim that DBT is a software technique to improve the performance. VMEXITs lead to mode change between gust and hypervisor mode. It is one of the main overhead in virtualization. To avoid a number of VMEXITs by clustering instructions, they also use DBT.

In this paper, we argue that DBT is very attractive and essential technique to full virtualization with lower overhead. In addition, there is no processor with the hardware assist for virtualization in space sector yet. Therefore DBT is one of the main cores to achieve full virtualizing spacecraft computer and improve performance as well.

4.1 The overall structure

DBT in the prototype translates SPARC v7 instructions for ERC32 to SPARC v8 instructions for LEON4 on-the-fly. We translate the guest image into dynamic basic blocks. The control flow of basic blocks is determined at runtime. They are started right after control transfer instructions such as branch, and then finished at next control transfer instructions. The DBT supports code cache to reduce translation overhead. If the block has been executed before, DBT can skip translation step by jumping to code cache directly. Intermediate Representation (IR) step is unnecessary in our case. It is because our guest and the host processor have almost the same instruction set architecture (ISA).

4.2 Classification of instructions

As shown in Table 1, we classify SPARC v 7 instructions into three categories: privileged, sensitive, and normal instructions. Privileged and sensitive are critical instructions accessing system resources such as processor status register or memory. In this classification, privileged instructions cause traps and sensitive instructions do not generate traps when the host processor is in user mode. Load and store instructions are typical sensitive instructions. We categorize control transfer instructions to sensitive instructions. Those instructions identify basic block units. Normal instructions do not belong to privileged and sensitive and it includes Arithmetic, logical, and shift operations.

4.3 Translation rules

Normal instructions. These instructions are the ones that can be executed without extra supervision by the virtualization layer. They are executed simply by copying the guest instruction.

Privileged instructions. DBT translates privileged instructions to other instructions according to following behaviors. We check whether guest code is in privileged mode or not via the processor state register (PSR) of guest's VCPU. When a guest is running in privileged mode, the translated privileged instructions are executed.

Unconditional branch instructions. Handling unconditional branches such as CALL, DBT calculates the guest's next PC, and then inserts calculated PC to the end of a translated block. Finally unconditional branches are removed.

Conditional branch instructions. Dealing with conditional branches, DBT inserts two guest's next PCs and adjusts offset of conditional branches. Register window moves instructions. The SAVE and RESTROE instruction move the register window of SPARC architecture. DBT needs to translate them to several instructions which mean that current window pointer (CWP) of guest's PSR is changed for moving the guest's register window.

Load and store instructions. DBT inserts instructions to calculate the correct address of operands.

Table 1 Classification of SPARC v7 ISA

Type	#	Instructions
Privileged	20	LDSBNA, LDSHA, LDUBA, LDUHA, LDA, LDDA, STBA, STHA, STA, STDA, STDFQ, LDSTUBA, SWAPA, RDPSR, RDWIM, RDTBR, WRPSR, WPWIM, WRTBR, RETT
Sensitive	more than 31	LDSB, LDSH, SDUH, LD, LDC, LDDC, LDCSR, STB, STH, ST, STD, STC, STDC, STCSR, SDSTUB, SWAP, LDD, LDF, LDDF, LDFSR, STF, STDF, STFSR, SAVE, RESTORE, Bicc, FBicc, CALL, JMPL, Ticc, FPop
Normal	28	ADD, ADDX, SUB, SUBX, ANDX, AND, ANDN, OR, ORN, XOR, XNOR, SLL, SRL, ADDcc, ADDXcc, TADDcc, TADDccTV, MULScc, SUBcc, ANDcc, ANDNcc, ORcc, ORNcc, XORcc, XNORcc, RDY, WRY, FLUSH

5. Experiment

The experiments were performed on dual-core LEON4 based GR-LEON4-ITX board from Aeroflex Gaisler[23]. The prototype is designed as loadable kernel module on Linux 2.6.21.1. We programmed a simple benchmark that has a subroutine to swap between two values. It has 22 sensitive instructions and 18 instructions are LD/ST among the sensitive instructions. While the simple benchmark program is executed, we measure translation and execution time respectively. For the reappearance of the trap-and-emulation, we modified *do_priv_instruction()* in Linux kernel. We confirmed that two values in memory are swapped using GDB.

Table 2 Total average time (in ms) with a loop for repeating the guest program once, 100 times, 1000 times.1

The number of repeated loop	1	100	1000
Trap-and-Emulate	7.8	443	5137.7
DBT	5.6	26.6	216.7

Table 3 Translation and execution time (in ms) when the guest program is executed once

	Translation	Execution	Total
Trap-and-Emulate	1.7	6.1	7.8
DBT	4.9	0.7	5.6

Table 2 shows results about the total average time for repeating ten executions independently. The measurement was performed using the benchmark program with a loop for repeating it once, 100 times and 1000 times. We distinguish between translation and execution time in our result. It is presented in Table 3. These tables show that trap-and-emulate version is slower than the DBT to emulate critical instructions at user level. As shown in Table 3, trap-and-emulate spent 6.1ms for the execution. It is because trap handling requires executing extra instruction. On the other hand the DBT took only 0.7ms for the execution even though it spent 4.9ms to translate the guest program. As we repeated the guest program, the total time could be shorter because of the code cache. We found the result that is wide variation of execution time due to trap. In experiment of repeated loop, best execution time is two times faster than worst case. In real-time system for spacecraft running on RISC machine, load and store instructions are used very frequently. They are usually sensitive instructions in RISC machines. We need to think about solutions for the optimization because our result shows the frequent traps or VMEXITs may cause of lager overhead.

6. Conclusion

In this paper, we developed a dynamic binary translator engine for spacecraft computer. Dynamic binary translator is the core the core technology for full virtualization and we need to reduce runtime overhead for space use. As our first step, we developed the dynamic binary translator, which reduce number of traps in translation step. Even though we tested with a small size ERC32 program, we could see potential for achieving full virtualization in space sector.

In the future work, we expect this attempt to become the important first step for full virtualization in space sector. Through experiment, we found that load and store instructions are used very frequently in spacecraft system running on RISC machine. Therefore, we will be able to find good approaches about load and store instructions to reduce the much overhead for full virtualizing spacecraft computer.

Reference

1) Martin, H., James, W., Knut, Eckstein., Maria, H., and Kjeld, H. 2012. ESA Roadmap for IMA Spin-in to Spacecraft Avionics. Data System In Aerospace (May. 2012). DASIA 2012. Dubrovnik, Croatia.

2) Aeronautical Radio Inc. 1991. Design Guidance for Integrated Modular Avionics. ARINC Report 651.

3) A. Crespo, I. Ripoll, M. Masmano, P. Arberet, and J.J. Metge. 2010. XtratuM: an Open Source Hypervisor for TSP Embedded

Systems in Aerospace. Data System In Aerospace (May. 2010). DASIA 2010. Istanbul, Turkey.
4) SYSGO AG. PikeOS RTOS Technology. http://www.pikeos.com
5) VMware white paper. 2007. Understanding Full virtualization, Paravirtualization, and Hardware Assist. WP-028-PRD-01-01 http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf
6) Popek, G. J., and Goldberg, R. P. 1974. Formal Requirements for Virtualizable Third Generation Architectures. Communications of the ACM, Volume 17, 412-421.
7) Keith Adams and Ole Agesen. 2006. A Comparison of Software and Hardware Techniques for x86 Virtualization. Proceedings of the 12th international conference on Architectural support for programming languages and operating systems (Oct. 2006). ASPLOS-XII. San Joes, California.
8) Ole Agesen, Jim Mattson, Radu Rugina, and Jeffrey Sheldon. 2012. Software techniques for avoiding hardware virtualization exits. Proceedings of the 2012 USENIX conference on Annual Technical Conference. USENIX ATC'12.
9) Abel Gordon , Nadav Amit , Nadav Har'El , Muli Ben-Yehuda , Alex Landau , Assaf Schuster , Dan Tsafrir. 2012. ELI: bare-metal performance for I/O virtualization. Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems (Mar. 2012). ASPLOS '12. London, England, UK.
10) Intel VT. http://ark.intel.com/Products/VirtualizationTechnology
11) Cortext A15. http://www.arm.com/products/processors/cortex-a/cortexa15.php
12) ERC32, http://www.atmel.com
13) LEON4, http://www.gailser.com
14) P. Barham, B. Dragovic, K. Fraser, S. Hand, T Harris, A. Ho, R. Neugebauer, I. Pratt and A.Warfield. 2003. Xen and the art of Virtualization. Proceeding of the nineteenth ACM symposium on Operating systems principles, 164-177. SOSP ’03. NEW York, NY.
15) A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. 2007. KVM: The Linux Virtual Machine Monitor. In proceedings of the Linux Symposium, 225–230.
16) Jon Watson. 2008. VirtualBox: bits and bytes masquerading as machines. Linux Journal, Volume 2008, Issue 166, February 2008
17) Aeronautical Radio Inc. 2005. Avionics Application Software Standard Interface (Part 1): Required Service. ARINC Specification 653P1-2.
18) RETMS, http://www.rtems.com
19) Swaroop Sridhar, Jonathan S. Shapiro, Eric Northup, Prashanth P. Bungale. 2006. HDTrans: an open source, lowlevel dynamic instrumentation system. Proceedings of the 2nd international conference on Virtual execution environments (Jun, 2006). VEE`12. Ottawa, Ontario, Canada.
20) Bruening, D., Garnett, T., and Amara-Singhe, S. 2003. An Infrastructure for Adaptive Dynamic Optimizations. Proc. nternational Symposium on Code Generation and Optimization. 265-275. CGO '03.
21) Luk, C. K., Cohn, R. S., Muth, R., Patil, H., Klauser, A., Lowney, P. G., Wallace, S., Reddi, V. J., And Hazelwood, K. 2005. Pin: Building Customized Program Analysis Tools With Dynamic Instrumentation. Programming Languages Design and Implementation (Jun. 2005), 190-200. PLDI '05.
22) H, Joe., H, Jeong., Y. Yoon., H. Kim., S. Han., and H. Jin. 2012. Full Virtualizing Micro Hypervisor for Spacecraft Flight computer. Digital Avionics System Conference (Oct. 2012). DASC 2012. Williamsburg, VA.
23) LEON4 Itx board - Aeroflex Gaisler http://www.gaisler.com/index.php/products/boards/gr-leon4-itx

Acknowledgments This research was supported by National Space Lab Program through the National Research Foundation (NRF) of funded by the Ministry of Education, Science and Technology.(NRF-2011-0020905)