UKCF: A New Graphics Driver Cross-Platform Translation Framework for Virtual Machines

Haitao Jiang[1], Yun Xu[1], Yin Liao[1], Guojie Jin[2], and Guoliang Chen[1]

[1] School of Computer Science and Technology,
University of Science and Technology of China, Hefei, China
[2] Institute of Computing Technology Chinese Academy of Sciences, Beijing, China
jhtjht1@mail.ustc.edu.cn

Abstract. Virtual machine with dynamic binary translation system is the key technology to solve software compatibility problem. But traditional user space binary translation systems can't translate hardware drivers such as graphics drivers in operating system kernel directly, instead, they need translate the entire operating system. To solve this problem, we designed a new binary translation framework. This framework has a user space translator and a kernel space translator working coordinated and can translate graphics drivers directly. Compared with traditional binary translation systems, this framework can significantly improve the performance of the virtual machine. Based on our experiment, the multimedia performance of virtual machines can been improved about 30%.

Keywords: virtualization, binary translation, operating system, driver, cross-platform.

1 Introduction

At present, with the innovation of computer architecture, Software compatibility issue has become increasingly prominent. Virtual machine (VM) can run binary format software on different architectures without modifying of source code, and thus become an important technology to solve this problem[1][7]. Graphic driver translation is a key problem in software compatibility issue because most multimedia softwares need graphic drivers working with them.

The key technology to run software on different architectures is binary translation which can translate an instruction stream based on one ISA (Instruction System Architecture) into the corresponding instruction stream based on another ISA[2]. There are two kinds of binary translation systems: static binary translation system and dynamic binary translation system. Interactive virtual machines always use dynamic translation system, which translates instructions dynamically during execution of programs.

Traditional virtual machines with dynamic binary translation run upon host operation system (such as VMware[3], QEMU[5][6], virtualbox[4]), these virtual

J.J. Park et al. (Eds.): NPC 2012, LNCS 7513, pp. 156–163, 2012.

machines have some advantages: clean hierarchy, easy to use, easy to migrate from one compute to another. But, they can't translate hardware drivers such as graphic drivers in operating system kernel directly; instead, they need translate the entire operating system[3][4][5]. This kind of translation mode brings about significant additional time consuming. At present, large-scale multimedia software usually need huge amount of computing resources and very high performance requirement, so this kind of low efficiency translation mode usually can't meet the demand.

To solve this problem, we designed a new binary translation framework (User space translator and Kernel space translator Cooperate Framework, UKCF). This framework has two translators, one works in user space, and the other works in operating system kernel space to translate the graphic driver directly. Compared with traditional binary translation systems, UKCF needn't translate the instructions of the entire operating system; instead, it only need translate the instructions of the application software's operating system kernel module (such as a graphic driver). UKCF can significantly improve the performance of the virtual machine because it reduces the number of instructions which need to be translated.

The remainder of this paper is as follows. In section 2, we introduce existing widely used virtual machines, discuss why they can't translate graphic driver directly. In section 3, we introduce the design and structure of UKCF. The experiment results and analysis are drawn in section 4.

2 Existing Technology

VMware is a widely used business virtual machine. It provides an abstraction of x86 PC hardware to run multiple operating systems at the same time. As a mature business system used by millions of users, VMware has high stability and efficiency. But, it has no dynamic binary systems and can't run application softwares on different ISA, such as MIPS. So it can't be used to solve the graphic driver translation problem.

Virtualbox is a powerful x86 virtualization software for enterprise and home use which is freely available as Open Source Software under the terms of the GNU General Public License (GPL). It has comparative performance with VMware. It also has no dynamic binary systems, and is not a solution to graphic driver translation problem.

QEMU is a multihost, multitarget virtual machine. It can run on multiple host ISA, such as X86, X86-64, MIPS, PowerPC and so on, and it can emulate multiple guest ISA too[6]. So it can be used to resolve software compatibility problem. QEMU runs upon host operating systems and it can only translate the instructions in user space. Graphic drivers are embedded in operating system kernel space, so it can't translate the instructions of graphic drivers directly. Fig. 1 gives a clear view of this problem.

Kernel-based Virtual Machine (KVM)[9], is a subsystem of Linux operating system which leverages virtualization extensions of commodity x86 processors to add a virtual machine monitor capability to Linux. Using KVM, multiple virtual machines can run on Linux operating system. This is an operating system level virtualization

system which can run in kernel space. But it has no dynamic binary translation module so can't be used to solve the graphic driver translation problem.

From fig. 1 shows that, to translate graphic drivers directly, we need to translate and execute the instructions of graphic drivers in operating system kernel space (as fig. 2 shows). In view of this point, we designed UKCF, a new binary translation framework with an operating system kernel space translator to solve this problem.

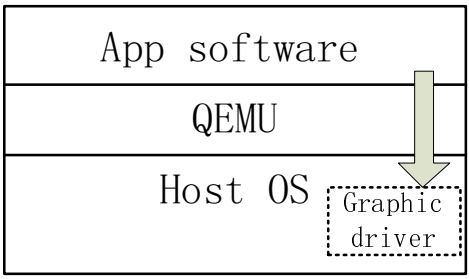

Fig. 1. The locations of QEMU and graphic drivers

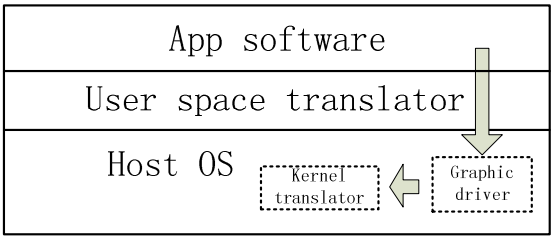

Fig. 2. The locations of graphic drivers and kernel translator

3 Design of UKCF

3.1 Workflow of Traditional Graphic Drivers

Graphic drivers are embedded into operating system kernels. When an application software needs to use the graphic driver, it first calls an operating system kernel API and traps into the kernel, and then call the functions of the graphic driver. During the execution of the graphic driver, it may call other operating system kernel functions. Fig. 3 shows the workflow of graphic drivers.

Two calling processes of fig. 3 must be handled by UKCF, one is the calling process from the kernel API to the graphic driver, and the other is the calling process from the graphic driver to the operating system kernel. When a kernel API calls a function of the graphic driver, UKCF needs intercept the calling action and start the translation mechanism to execute the instructions of the graphic driver. When the

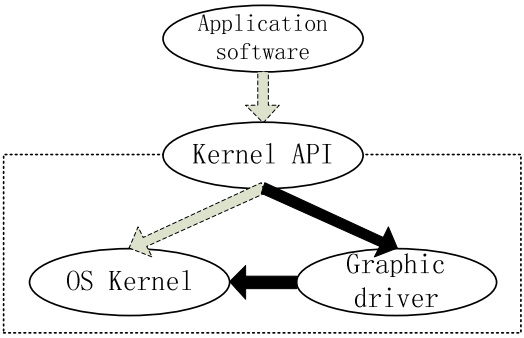

Fig. 3. The workflow of graphic drivers

graphic driver calls a function of the operating system kernel, UKCF needs end the translation mechanism and give the execution control to the operating system kernel. The detail method of this switching will be introduced in next section.

3.2 Workflow of the Graphic Driver in UKCF

In UKCF, instructions of the graphic driver and instructions of the operating system kernel have incompatible ISA, so, traditional kernel module loading method[8] can't embed the graphic driver into the kernel. To embed the graphic driver into the kernel, we treat the graphic driver as a stream of data and use an array in the operating system kernel to store it. When a function of the graphic driver is called, UKCF will get the address of the function, translate and execute the instructions of the function.

During the application software's execution, only a part of the instructions needs to be translated, and the other instructions must be executed directly. To solve this problem, UKCF needs to do two additional works, one is monitoring entrance points and exit points, and the other is saving and loading the translation context.

When the operating system kernel calls a function of the graphic driver, the function's address is an entrance point. We use function shell technology to catch the entrance point. To work with the operating system kernel, the graphic driver must register its functions to the kernel[8]. To catch the functions' calling time, UKCF doesn't allow the graphic driver register its functions to the kernel directly, instead, UKCF register the shelled functions of the graphic driver to the operating system kernel. Fig. 4 shows this technology.

The shell of a function is another function, it is registered to the calling point of the shelled function. So when the kernel calls the shelled function, the shell function is called. When the shell function is called, it sends the address of the shelled function to UKCF, UKCF loads the translation context and starts the translation. The shell function also saves the return address of the shelled function, and when the shelled function ends, UKCF save the translation context and return to this address. This solves the entrance point monitoring problem.

UKCF monitors all the jump instructions' target address, if the target address is not in the content of the graphic driver, this means the target address is a kernel function. UKCF changes the pc to the target address and gives the control to the kernel. This solves the exit point monitoring problem.

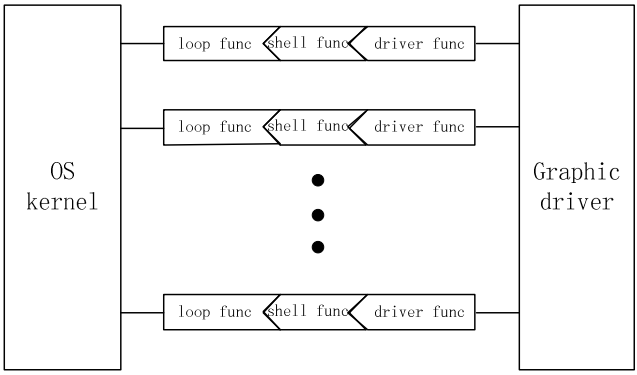

Fig. 4. The shelling technology of UKCF

3.3 Structure of UKCF

Fig. 5 shows the structure of UKCF. It contains six modules: a user space translator, a kernel space translator, a graphic driver shell, a jump monitor, and two CPU simulators. The user space translator is the first starting module of UKCF, it loads the application software and translates it. CPU simulator simulates a guest ISA CPU to execute the instructions. In UKCF, the CPU simulator is the same as traditional virtual machines[5]. The graphic driver shell monitors the entrance point of the graphic driver and gives the entrance address to the kernel space translator. The

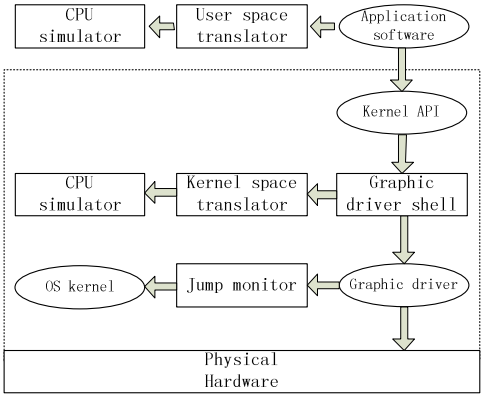

Fig. 5. The structure of UKCF

kernel space translator works together with the jump monitor, when the jump monitor catches a call from the graphic driver to the operating system kernel, the kernel space translator stops translation, saves the translation context, modifies the pc and gives control to the operating system kernel.

4 Experiment and Analysis

Traditional user space binary translation systems can't translate graphic drivers in operating system kernel directly, instead, they need translate the entire operating system. UKCF can translate graphic drivers directly in kernel space, so the number of instructions needing to be translated is reduced. To analyse the performance of UKCF quantitatively, we defined the concept of performance loss ratio (*PLR*), for a given instruction stream *i* with execution time t_1 on its original ISA platform, if its execution time on a different ISA platform with a binary translation system is t_2, then the instruction stream PLR_i of the binary translation system is:

$$PLR_i = t2/t1 \tag{1}$$

The PLR of the binary translation system is the average of the PLR_i:

$$PLR = \sum_{i=1}^{n} PLR_i/n \tag{2}$$

In order to facilitate the presentation, we call the traditional translation mode (translating the entire operating system) scheme 1, call the UKCF's translation mode (translating the graphic driver directly) scheme 2. We use NUM_1 to identify the number of instructions needing to be translated by scheme 1, NUM_2 to identify the number of instructions needing to be translated by scheme 2, *t* to identify the average execution time of one instruction, $TIME_1$ to identify the execution time of scheme 1, $TIME_2$ to identify the execution time of scheme 2, then:

$$TIME_1 = t \times NUM_1 \times PLR \tag{3}$$

$$TIME_2 = t \times NUM_2 \times PLR + t \times (NUM_1 - NUM_2) \tag{4}$$

So the performance improved percentage of UKCF is:

$$P = (TIME_1 - TIME_2)/TIME_1 = \frac{(NUM_1 - NUM_2)(PLR-1)}{NUM_1 \times PLR} \tag{5}$$

Formula 5 shows a fact that, the execution time saved depends on the reduction percentage of the instructions needing to be translated.

We tested the reduction percentage of the instructions needing to be translated, the host ISA platform is the Loongson 3A platform[10][11], which has a MIPS compatible ISA. The host operating system is Linux, the guest ISA is x86 IA-32. The compared binary translation system is QEMU, which has been proven to be a very fast dynamic binary translation virtual machine[6]. We used mplayer[13] and a group of videos as the testing set, the results are shown by fig. 6:

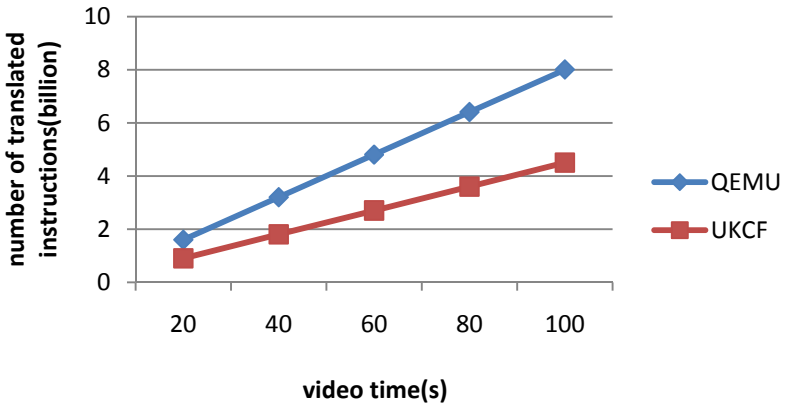

Fig. 6. Experimental results: the numbers of translated instructions of QEMU and UKCF

Fig. 6 shows that, UKCF only needs to translate about 60% instructions. In our experiment environment, the PLR of QEMU is about 6. We can compute the performance improved percentage of UKCF using formula 5, the result is about 33%. We tested the result based on real videos played by mplayer, the experiment results are shown by fig. 7:

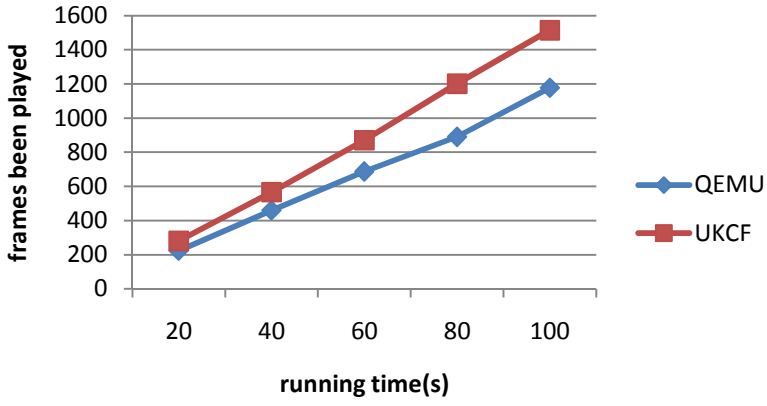

Fig. 7. Experimental results: the performance compare of QEMU and UKCF

From fig. 7 we can see that, during the same time, UKCF can play about 30% more frames than QEMU, this result confirms the analysis based on formula 5.

5 Conclusion

This paper designed a new graphics driver cross-platform translation framework named UKCF for virtual machines. Compared with existing dynamic binary

translation system needing translate the entire operating system to run the graphic driver, UKCF can translate and execute the graphic driver in operating system kernel directly. Experiment results improved that UKCF runs faster than existing binary translation systems about 30%.

How to improve the performance is the key problem of binary translation systems. How to make good use of the advantages of kernel space, such as memory allocation privilege and direct hardware access privilege, to further improve the performance of UKCF, is the future research goal.

References

1. Smith, J.E.: A unified view of virtualization. In: Proceedings of the 1st ACM/USENIX international Conference on Virtual Execution Environments (June 2005)
2. Sites, R.L., Chernoff, A., Kirk, M.B., et al.: Binary translation. Communications of the ACM CACM Homepage Archive 36 (1993)
3. http://www.vmware.com
4. http://www.virtualbox.org
5. http://wiki.qemu.org
6. Bellard, F.: Qemu, a fast and portable dynamic translator. In: Proceedings of the USENIX 2005 Annual Technical Conference, pp. 41–46 (2005)
7. Altman, E.R., Kaeli, D., Sheffer, Y.: Welcome to the Opportunities of Binary Translation. IEEE Computer 33 (2000)
8. http://www.linux.org
9. Kivity, A., Kamay, Y., Laor, D., Lublin, U., Liguori, A.: kvm: the Linux virtual machine monitor. In: OLS 2007: The 2007 Ottawa Linux Symposium, pp. 225–230 (July 2007)
10. Hu, W.-W., Wang, J., Gao, X., et al.: Godson-3: A Scalable Multicore RISC Processor with x86 Emulation. IEEE Micro. 29, 17–29 (2009)
11. Hu, W.-W., Wang, J., Gao, X., et al.: Micro-architecture of Godson-3 Multi-Core Processor. In: Proceedings of the 20th Hot Chips (2008)
12. http://www.spec.org
13. http://www.mplayerhq.hu