KBT: Operating System Kernel Level Binary Translation System

Haitao Jiang
School of Computer Science and Technology, University of Science and Technology of China, Hefei, China.
jhtjht1@mail.ustc.edu.cn

Yun Xu*, Yin Liao, Guojie Jin, Guoliang Chen
School of Computer Science and Technology, University of Science and Technology of China, Hefei, China.
Institute of Computing Technology Chinese Academy of Sciences, Beijing, China.
xuyun@ustc.edu.cn

***Abstract*—with the diversification of hardware platforms, software compatibility issue has become increasingly prominent. Virtual machine with dynamic binary translation system is the key technology to solve this problem. This paper designs an operating system kernel level virtual machine with binary translation systems (KBT) which is embedded into kernel space as a kernel module. KBT reduces the number of virtual layers of the computer system, and introduces further optimization strategies using kernel mode advantages. Experiment results improves that, KBT's efficiency is higher than existing binary translation systems about 25%. KBT also has some interfaces to support kernel module translation systems to run complex software, which are hard to run upon existing user space virtual machines.**

***Index Terms*—virtual machine, binary translation, operating system, kernel space, kernel module, virtual layer.**

I. 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 any modification of source code, thus becomes the key technology to solve this problem[1].

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 computer hardware and software systems can be divided into different virtual layers: hardware layer, operating system layer and application layer(Fig.1 (a)). A virtual machine which run an entire operation system (such as VMware[3], QEMU[5][6], virtualbox[4]) adds two additional virtual layers into the computer system(Fig.1 (b)): virtual machine layer and guest operation system layer, each virtual layer causes additional performance loss. On the other hand, a virtual machine with dynamic binary translation systems needs translate instructions during the execution of the program, and maintains runtime instructions management mechanism which causes large number of performance overhead. With the increasing of software systems' complexity, how to reduce the performance overhead of dynamic translation systems becomes an important issue[4].

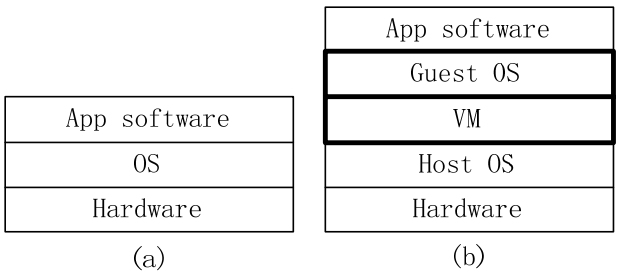

Figure 1: computer systems' virtual layers. The left figure shows virtual layers of computer systems without any virtual machine, the right figure shows virtual layers of computer systems with virtual machines.

Traditional virtual machines with dynamic binary translation systems run upon host operation systems (such as widely used open source software QEMU), these virtual machines have some advantages, such as, clean hierarchy, easy to use, easy to be migrated from one compute to another. But, because of the virtual layers added by these VMs, the entire computer system's performance is reduced significantly. For high-performance applications, such as complex interactive multimedia applications and large-scale scientific computing software, these VMs usually fail in performance. At the same time, some complex software may have modules embedded in operating system kernel, such as the hardware driver modules of multimedia players. Traditional user level virtual machines are hard to run this kind of software.

To resolve this problem, we considered lowering the layer of the virtual machine, and merging the virtual machine with host operating system. The virtual machine with dynamic binary translation systems becomes a

*Corresponding author.

doi:10.4304/jsw.8.6.1275-1280

kernel module of the operating system, and runs in kernel mode. This strategy can significantly improve the performance of the entire computer system, and can also support kernel module translation systems which translate software's kernel modules. This paper designed an operating system kernel level virtual machine with binary translation systems (KBT), KBT is embedded into operating system kernel space as a kernel module, and only a simple boot module stays in user space which boots and passes information to operating system kernel. KBT reduces the number of virtual layers of the computer system with virtual machines (Fig. 2). At the same time, further optimization strategies using kernel mode advantages were introduced to improve the performance of the computer system. We also designed some interfaces to support kernel module translation systems to run complex application software.

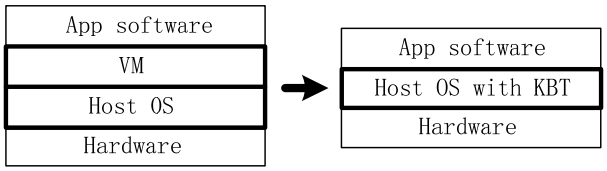

Figure 2: virtual layers of a computer system with a virtual machine. The left figure shows virtual layers of user space virtual machines, the right figure shows virtual layers of operating system kernel space virtual machines.

The remainder of this paper is as follows. In section II, we introduce existing widely used virtual machines, discuss their advantages and disadvantages. In section III, we introduce the design and structure of KBT. The experiment results and analysis are drawn in section IV.

II. Existing Virtual Machines

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 business system used by millions of users, VMware has high stability and efficiency. But, because it has no dynamic binary system, it can't run operating systems on different ISA, such as MIPS. So it can't be used to solve the software compatibility problem.

Virtualbox is a powerful x86 virtualization software which is freely available as an 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, so is not a solution to software compatibility problem.

QEMU is a multi-host, multi-target 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 is an Open Source Software and runs upon host operating systems. As discussed before, this kind of virtual machines add additional virtual layers to computer system (Fig. 1), translate instructions during the execution of the program, and maintain runtime instruction management mechanism, so their efficiency are not very high and can't run complex software smoothly. At the same time, as a user space virtual machine, QEMU can't run applications with kernel modules.

DigitalBridge[8] is a binary translation system which can also run on different host ISA, and can emulate different guest ISA. It is also a user space binary translator and has similar mechanism with QEMU. How to optimize the efficiency and how to translate kernel modules are problems needed to be resolved.

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 software compatibility problem.

The relationship of above virtual machine systems are shown by Fig. 3. From Fig. 3 we can see that KBT is the only virtual machine running in kernel space which can handle software compatibility problem, and KBT is also the only system which can run complex applications with kernel modules.

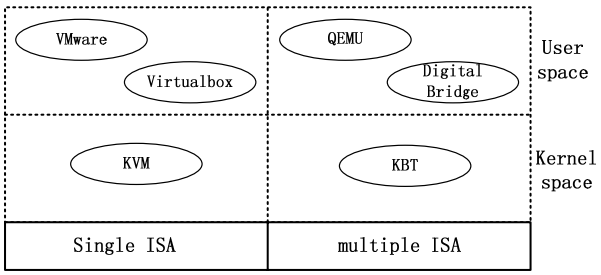

Figure 3: classification of virtual machines

III. Design of KBT

A. Advantages of Kernel Space

Modern operating systems have two run levels: user space and kernel space, corresponding to different CPU privilege levels. Take x86 CPU for example, it has four privilege levels from 0 to 3, when a process runs in user space, the CPU runs on privilege level 3, and when a process runs in kernel space, the CPU runs on privilege level 0. Different privilege levels have different restrictions, for example, the IN and OUT instructions used to input and output information to hardware ports can be used on privilege level 0, but can't be used on privilege level 3. Normally, only the operating system kernel runs in kernel space, other processes run in user space. When a process running in user space wants to communicate with hardware ports, it must request an operating system service and trap into the kernel. Existing virtual machines with dynamic binary translation systems run in user space, such as QEMU.

Compared with processes running in the user space, kernel space processes have many advantages in efficiency. Such as, for a virtual machine running in the

user space, the guest process' kernel requests must be encapsulated by the virtual machine, and then trap into the kernel, this usually has significantly performance loss. Modern operating systems support the multiple processes schedule mechanism, user space virtual machines can only take limited time slices allocated by the operating system to run guest processes. Of cause, for common users, multiple processes schedule mechanism can bring about significantly convenience. But when we use a virtual machine to run guest operating systems, we always don't need other processes except the virtual machine itself. In this case, kernel space virtual machines have significant advantages. For some complex applications' kernel modules, such as multimedia players' 3D hardware accelerating modules, the translation can only be done in kernel space. User space virtual machines are hard to handle this problem.

B. Structure of KBT

Based on the above discussion, we designed an operating system kernel level dynamic binary translation system (KBT). KBT has five main modules: User Monitor, CPU Simulator, Translator, Hardware Adapter and Kernel Monitor. User Monitor boots guest applications, and manages the interactive between the user space and the kernel space. CPU Simulator is the core module of KBT, it simulates a guest computer system's CPU. Kernel Monitor provides the ports for kernel module translation systems to translate the complex applications' kernel modules. Hardware Adaptor transmits the hardware operations of the guest software to physical hardware. The structure of KBT is shown by Fig. 4.

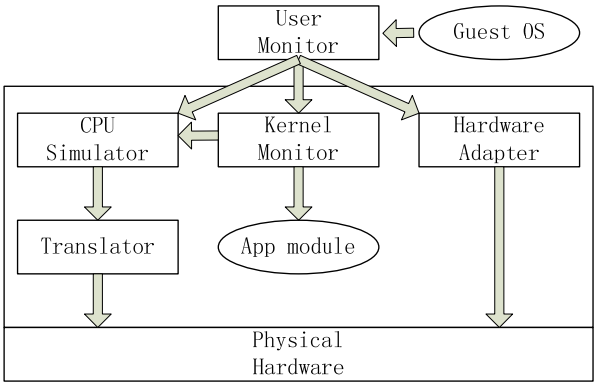

Figure 4: structure of KBT

User Manager is the first launched module of KBT, it boots the guest software in user space, and handles the interactive between user space and kernel space. Take Windows operating system's simulating for example, User Manager boots the Windows image into memory, and sends the memory location of Windows image to CPU Simulator, then gives control to CPU Simulator. When Windows needs handling some user space requests, such as user interactions, User Manager captures the request and sends it to CPU Simulator.

CPU Simulator simulates a guest computer system's CPU, and coordinates the entire KBT system's work. It's the core function module of KBT. CPU Simulator needs simulating the modern CPU's instruction execution function, instruction cache function, privilege level management function and so on. It also integrates the memory manage unit's function, such as virtual memory mapping. The structure of CPU Simulator is shown by Fig. 5. CPU Simulator has Register Issue Engine, Memory Engine and Environment Engine coordinated by Execution Engine. Register Issue Engine provides virtual registers to Execution Engine, Memory Engine provides virtual memory space, and provides virtual memory management mechanism. Environment Engine provides context switch function and stack function needed by the execution of a process.

The registers of CPU can be divided into general registers and control registers. General registers are used to store arithmetic's operands and results, control registers are used to control CPU's execution status, such as privilege levels switch. Register Issue Engine uses General Register Simulator and Control Register Simulator to simulate the two kinds of registers. Take x86 32 bits CPU for example, General Register Simulator simulates eight general registers: EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI. Execution Engine can use these general registers by the corresponding variables. Correspondingly, Control Register Simulator simulates the control registers of x86 CPU, such as CR control registers. Control Register Simulator also simulates the flag registers such as EFLAGS registers. Physical Register Manager maps the virtual registers to real memory cells or real registers. Compared with general registers, control registers are not used very frequently, so they are mapped to memory cells to lower the pressure of physical registers. But for the frequently used general registers, mapping them to the memory cells will bring about serious memory access overload. So, Physical Register Manager maps them to the physical registers. Liao gave a strategy to solve this kind of problem in [10].

Memory Engine provides virtual memory view to Execution Engine. All of the memory access's addresses used by Execution Engine are not physical memory addresses, but virtual memory addresses, and need to be translated to physical addresses by Memory Engine. Just to be clear, the physical addresses talked here are not the physical addresses of real physical computers, they are physical addresses of virtual computer system provided by the virtual machine. Software can't operate real physical memory addresses of computers. Virtual memory view can bring great benefits to virtual machine design, and it provides the possibility of running whole Operating Systems and Distributed Network Systems on virtual machines. Virtual Memory Manager Module supports 4G virtual memory, and supports page based virtual memory management. Virtual TLB simulates a Translation Lookaside Buffer (TLB) to accelerate the virtual address translation. Virtual TLB is a hash table storing physical pages. It uses hash values of virtual addresses' page numbers to index the hash table. The search of physical pages can be accomplished in constant time like real physical TLB. The physical addresses are

sent to Translator before instruction translation, the addresses of the back-end instructions are already physical addresses, so this translation of virtual memory addresses doesn't consume the time of back-end instructions' execution. Physical Memory Manager allocates physical memory for Virtual Memory Manager, and maps the virtual memory to physical memory allocated. It doesn't allocate all the memory used by KBT at once, but allocates memory based on the requirement. This strategy can lower the memory pressure of the computer system.

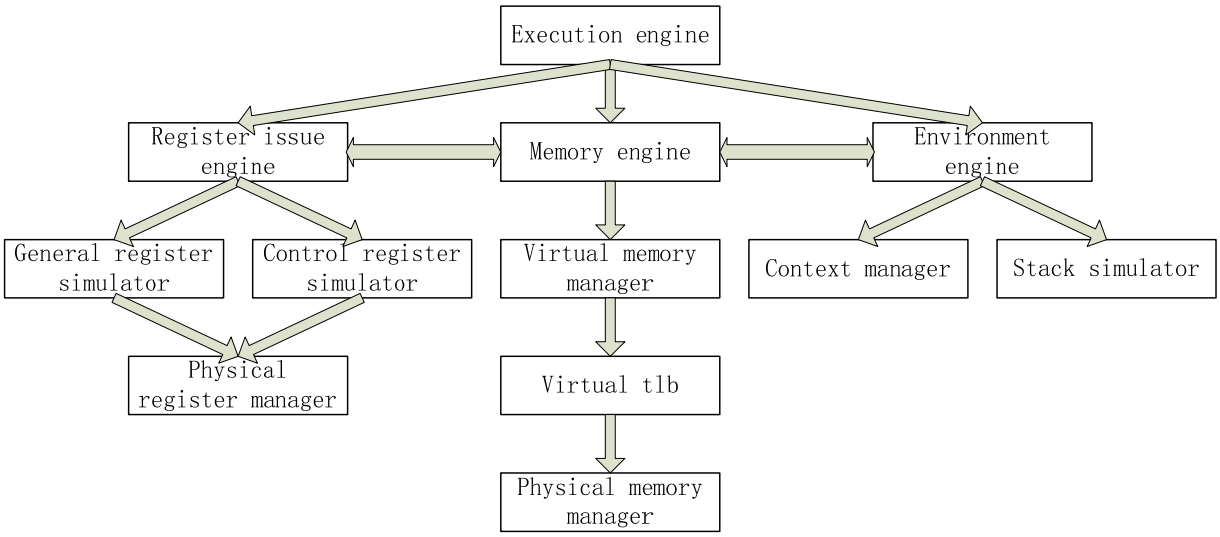

Figure 5: structure of CPU Simulator

Environment Engine provides the process execution environment, such as the context and stacks. Compared with other applications, the virtual machine with dynamic binary translation system has a unique feature that guest processes and the virtual machine process are in the same process context. So, context switching and saving mechanism is needed to protect process contexts of guest processes and the virtual machine process. Context Manager is the module handling this problem. Stack Simulator provides the stack mechanism to support procedural programming applications.

Translator translates the guest ISA instructions into host ISA instructions to run them on the host computer. It can be regarded as a special "Just In Time" compiler, the guest instructions are the input data, and the host instructions are the output data. Translator logically maintains a mapping table from guest ISA instructions to host ISA instructions. When CPU Simulator sends a guest ISA instruction to Translator, Translator searches this mapping table, finds the corresponding host ISA instructions. In KBT, the searching procedure of this mapping table can be accomplished by decoding guest ISA instructions which can be accomplished in constant time. The execution efficiency of host ISA instructions generated by Translator has important influence to the performance of the whole virtual machine, so the mapping table must be optimized as far as possible to ensure that the host ISA instructions' execution is quite fast. KBT takes advantages of some preliminary artificial optimizations, deeper optimizations considering the dynamic compile technology will be a further research direction.

Besides translating instructions, the other important function of Translator is managing the translated host ISA instructions to reduce duplication translation and accelerating the instruction fetching speed of the virtual CPU. KBT uses Level based Instructions Indexing Strategy (LIIS)[11] to manage the back-end host ISA instructions. Based on the special locality of the instructions, this strategy uses targeted replacement algorithms to cache the back-end host ISA instructions, significantly reduces the overhead of the instruction fetching.

Some complex software has kernel modules, such as multimedia player's 3D accelerate engine. This kind of software is widespread in computer systems. Existing user space virtual machines are hard to support this kind of software. KBT provides the possibility of solving this problem by the module named Kernel Monitor. Kernel Monitor monitors all the application's calls to kernel and provides interfaces to support kernel module translation systems(Fig. 6).

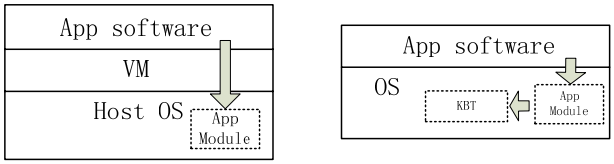

Figure 6: kernel module translation. The left figure shows why user space virtual machines are hard to translate operation system kernel modules, the right figure shows that KBT can handle this problem.

Besides CPU and memory, the PCI hardware devices also impact the performance of computer systems. For example, for large 3D games, the performance of the video card is even more important than CPU. Existing user space virtual machines running upon the operation system are hard to operate physical hardware devices directly. They usually need simulate the physical

hardware by software method, and use kernel calls to operate the real physical hardware devices. This method brings serious performance loss. KBT can operate hardware devices directly because it runs in kernel space (Fig.7). Hardware Adapter translates the guest application's hardware operations to corresponding physical hardware operations. If the guest operating system has the device driver of the real physical hardware device (common operating systems always have), then no extra work would be done. Compared with the simulating method used by user space virtual machines, this method can improve the performance of the whole system significantly.

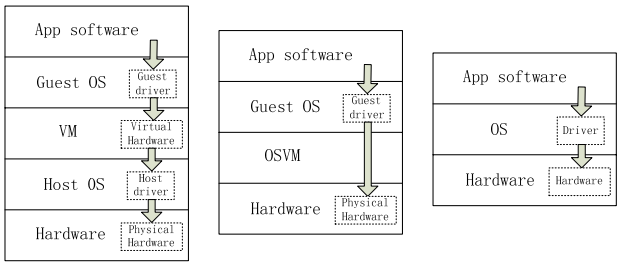

Figure 7: physical hardware virtualization. The first figure shows the physical hardware virtualization of user space virtual machines, the second figure shows the physical hardware virtualization of operating system kernel space virtual machines, the third figure shows the hardware in real computer systems.

IV. Experiments and Analysis

We tested the time performance of KBT on the Loongson 3A CPU[12][13], the host operating system was Linux, the guest ISA was x86 IA-32, host ISA was MIPS. The compared virtual machine system was QEMU, which has been proven to be a very fast dynamic binary translation virtual machine, the performance compare between QEMU and other virtual machines can be found in [6]. The test set was spec2000[14]. Fig. 8 shows the experiment results. In this experiment, KBT used the same instruction translation rules as QEMU. From Fig. 9 we can see that, the performance of KBT is higher than QEMU about 20%.

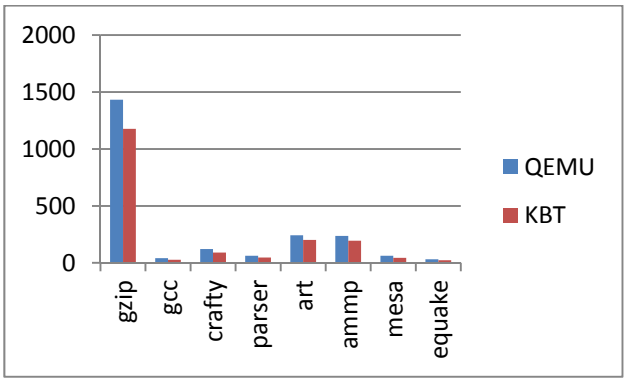

Figure 8: performance experiment of KBT

To further optimize the performance of KBT, we tested and analyzed the time percentages of KBT's different modules. The experiment results are shown by Fig. 9. From Fig. 9 we can see that, the time percentage of CPU Simulator is more than 80%. CPU Simulator is the bottleneck of KBT, needs depth optimization. The next one is Translator, which takes more than 10% time percentage. Hardware Adapter takes only about 5% time percentage, because spec2000 doesn't use hardware devices frequently. But for some special software, such as large 3D games, hardware devices, especially the video card, will take much more time percentage. Based on our experiment using mplayer[15] as testing set, the time percentage of Hardware Adapter raised to more than 15%.

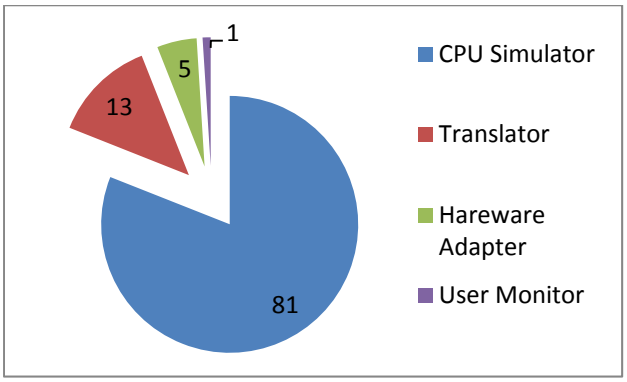

Figure 9: time percentages of KBT's different modules

Based on the analysis of section 3-B, back-end instructions generated by Translator have important affection to the efficiency of KBT. So, we tested the time proportion of the back-end instructions' execution. Results are shown by Fig 10.

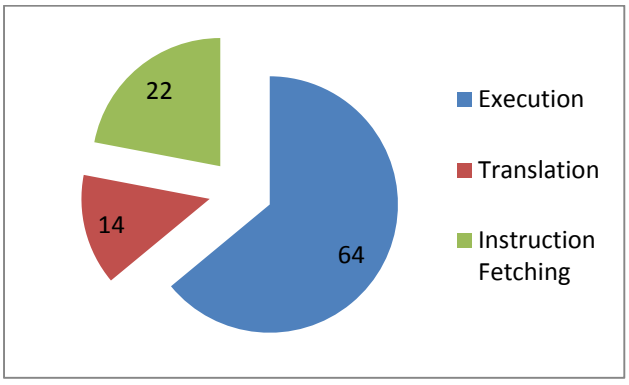

Figure 10: time percentage of the back-end instruction execution

Fig. 10 shows that, the time percentage of back-end instructions' execution is about 65%, so the quality of back-end instructions is the key factor of the system's efficiency. To quantitatively analyze the Translator's translation quality, we define the concept of Instruction Expansion Rate (IER). For the i-th guest ISA instruction block with p instructions, if the corresponding host ISA instruction block has q instructions, then we say:

$$IER_i = p/q \quad (1)$$

We use n to identify the number of instruction blocks, use m_i to identify the execution number of the i-th instruction block, and then the IER of Translator is the weighted average of IER_i:

$$IER = \frac{\sum_{i=1}^{n} IER_i * m_i}{\sum_{i=1}^{n} m_i} \quad (2)$$

The translation equality increases with the IER decreases. Based on our experiment, the IER of QEMU is 7.82. To improve the efficiency of KBT, we optimized the translation rules of KBT. After our optimization, the IER of KBT was reduced from 7.82 to 5.77, the efficiency was improved about 5%. After this optimization, the efficiency of KBT is higher than QEMU about 25%. Experiment results are shown by Fig 11.

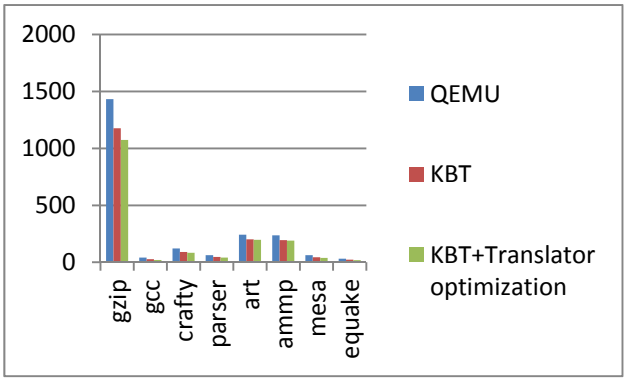

Figure 11: the performance experiment of KBT after preliminary optimization

IV. Conclusion

This paper designed an operating system kernel level virtual machine with binary translation systems (KBT) which can be used to solve software compatibility problem. Compared with existing user space virtual machines, running in kernel space brings many advantages to KBT. Experiment results improved that KBT runs faster than existing virtual machines. We also designed some interfaces to support kernel module translation systems to run complex software.

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

Acknowledgment

This work was supported by National Natural Science Foundation of China (Grant No. 61033009).

References

[1] James E. Smith. “A unified view of virtualization,” Proceedings of the 1st ACM/USENIX international Conference on Virtual Execution Environments, June 2005.
[2] RL Sites, A Chernoff, MB Kirk, et al. “Binary translation,” Communications of the ACM CACM Homepage archive, vol. 36, 1993.
[3] http://www.vmware.com
[4] http://www.virtualbox.org
[5] http://wiki.qemu.org
[6] Fabrice Bellard. “Qemu, a fast and portable dynamic translator,” In Proceedings of the USENIX 2005 Annual Technical Conference, 2005, pp.41-46.
[7] Erik R. Altman, David Kaeli, Yaron Sheffer. “Welcome to the Opportunities of Binary Translation,” IEEE Computer vol. 33, 2000.
[8] Bai Tong-xin, Feng Xiao-bing, Wu Cheng-gang, et al. “Optimizing Dynamic Binary Translator in DigitalBridge,” Computer Engineering, vol. 31, 2005.
[9] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. “kvm: the Linux virtual machine monitor,” In OLS '07: The 2007 Ottawa Linux Symposium, July 2007, pp.225-230.
[10] Liao yin, Sun Guang-zhong, Jiang Hai-tao, et al. “All registers mapping method for X86 to MIPS binary translation,” Journal of computer applications.
[11] Jiang Hai-tao, Xu Yun, Liao Yin, et al. “Level Based Binary Translation System Back-end Instruction Indexing Strategy,” Journal of Chinese Computer Systems.
[12] Hu Wei-wu, Wang Jian, Gao Xiang, et al., “Godson-3: A Scalable Multicore RISC Processor with x86 Emulation,” IEEE Micro, vol. 29, 17-29, 2009, pp.17-29.
[13] Hu Wei-wu, Wang Jian, Gao Xiang, et al. “Micro-architecture of Godson-3 Multi-Core Processor,” In Proceedings of the 20th Hot Chips, 2008
[14] http://www.spec.org
[15] http://www.mplayerhq.hu