논문 2011-06-30

New Hypervisor Improving Network Performance for Multi-core CE Devices

Cheol-Ho Hong, Miri Park, Seehwan Yoo, Chuck Yoo*

Abstract : Recently, system virtualization has been applied to consumer electronics (CE) such as smart mobile phones. Although multi-core processors have become a viable solution for complex applications of consumer electronics, the issue of utilizing multi-core resources in the virtualization layer has not been researched sufficiently.
In this paper, we present a new hypervisor design and implementation for multi-core CE devices. We concretely describe virtualization methods for a multi-core processor and multi-core-related subsystems. We also analyze bottlenecks of network performance in a virtualization environment that supports multimedia applications and propose an efficient virtual interrupt distributor. Our new multi-core hypervisor improves network performance by 5.5 times as compared to a hypervisor without the virtual interrupt distributor.

Keywords : Secure CE device, Hypervisor, Embedded multi-core processor, Network performance

1. Introduction

Recently, system virtualization has been applied to consumer electronics (CE) such as smart mobile phones to solve operating system (OS) security problems arising from Internet malware [1-3]. The virtualization layer or hypervisor allows multiple virtual machines (VMs) to be consolidated simultaneously in a physical machine. In a virtualized system, even though a guest operating system (guest OS) may be compromised by malware, other guest OSs and the system OS can remain uninfected by the isolation service provided by the hypervisor; this service enables the availability of the entire system and improves reliability.

In addition to reliability, system virtualization can provide user extensibility for a consumer to install personal virtual machines for the user's own purposes. Consumer preferences have evolved to require multiple capabilities such as multimedia data play and Internet connectivity in a single device. By using a hypervisor and permitting the installation of personal virtual machines, device manufacturers can satisfy sophisticated consumer preferences with less effort. As recent smart phones or digital TV systems adopt general-purpose hardware such as ARM processors, personal virtual machines can be installed in these systems easily.

As mobile audio and video applications have become heavily CPU dependent, CE devices are migrating to the multi-core and multiprocessor systems-on-a-chip (MPSoC) platforms [4-6]. The CPU-dependent applications such as H.264 decoders require massive computing power for decoding complicated encoded frames. The most promising way to guarantee the

* Corresponding Author
Manuscript received : 2011. 06. 15.,
Revised : 2011. 08. 03., Accepted : 2011. 08. 26.
Cheol-Ho Hong, Seehwan Yoo, Chuck Yoo : Korea University
Miri Park : LG Electronics
※ This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MEST) (No. 2011-0029848).

quality-of-service of these applications is to select multi-core processors such as the ARM MPCore series [8]. Practically, some CE device vendors have adopted multi-core processors for their set-top-box and digital TV products to provide true 1080p resolutions and better responsiveness of digital TV interfaces.

Although the complexity of applications has led embedded multi-core processors to be a viable solution, research of the hypervisor for these processors has not been accomplished sufficiently. Xen on ARM, a representative hypervisor for CE, is targeting a single-core platform. Although other commercial hypervisors support embedded multi-core processors, their design and implementation are unknown [11]. As CE devices are migrating to multi-core platforms, hypervisor research for multi-core devices is becoming essential.

In this paper, we present the hypervisor design and implementation for multi-core CE devices. In particular, the hypervisor aims not only to guarantee the quality-of-service of media applications but also to improve network performance by fully utilizing the multi-core resources in interrupt processing. The multi-core hypervisor that this paper presents is based on the ARM11 MPCore processor [8], [9] and is extended from the single-core MobiVMM, which is a research hypervisor for soft real-time applications [15], [16]. The main contributions of this paper are as follows:

First, we provide the experience of adopting virtualization technologies to multi-core platforms. We concretely and comprehensively describe the details of the hypervisor design, including not only a multi-core, but also a physical memory, and I/O devices virtualization, and multi-core-related subsystems that support virtual machines in the hypervisor.

Second, we provide hypervisor technologies for multimedia CE devices such as digital TV systems, set-top-boxes, and smart phones. Our hypervisor guarantees the quality-of-service by prioritizing multimedia-related interrupt handling and providing a preemptive virtual machine scheduler in the multi-core environment.

Third, we improve the network performance of general-purpose guest OSs when a soft real-time virtual machine and general-purpose guest OSs are consolidated in a multi-core CE system. We have found that if the two kinds of domains are running simultaneously, the network performance of general-purpose domains is deteriorated. In this case, the soft real-time virtual machine acquires most of the CPU bandwidth of the first core to process soft real-time related interrupts. To solve this problem, we implement a new and efficient virtual interrupt distributor that enables elaborated interrupt handling by increasing the utilization of the multi-core. In addition, we suggest methods of how and when to enable or disable the functionality of the virtual interrupt distributor by profiling and monitoring systems loads. Our additional functionality is placed in the virtualization layer so that guest OSs do not have to install any instrumentation. Our new multi-core hypervisor improves the network performance by 5.5 times compared with the hypervisor without a virtual interrupt distributor.

The remainder of this paper is structured as follows: In section 2, we explain the background of CE virtualization and related work. In section 3, we illustrate how a multi-core hypervisor is designed. Section 4 describes multi-core-related subsystems. Section 5 proposes a method for improving network performance, and section 6 provides the evaluation results. Finally, we conclude in section 7.

2. Background and Related Work

In this section, we present a brief background on the virtualization architecture

for CE devices and related work.

2.1. Virtualization Architecture for CE Devices

A type 1 hypervisor that directly runs on a physical machine is appropriate for CE devices to guarantee maximum performance. Conversely, a type 2 hypervisor runs on a host OS and uses physical devices indirectly through the interfaces of the host OS; it is usually adopted in a desktop or server environment.

Fig. 1 illustrates the architecture of the type 1 hypervisor for CE devices. It has the following characteristics:

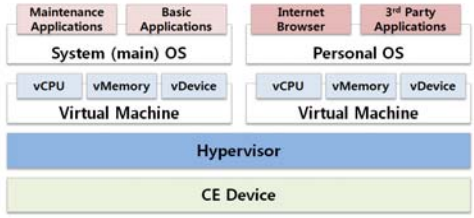

Fig. 1. Architecture of a type 1 hypervisor for CE devices

● At the bottom of the system software stacks, the hypervisor vir-tualizes all system resources such as a processor, a mem-ory, and I/O devices; it exports them in the form of a virtual machine.

● Each virtual machine provides a complete system environment, which is composed of virtual CPUs, a virtual memory, and virtual devices, to each guest OS.

● The number and composition of guest OSs can be variable according to the characteristics of the entire system and user applications. For example, the secure Xen on ARM [2] is composed of two identical Linux guest OSs: a secure guest OS for secure applications and an open guest OS for non-trusted applications. The OSTI architecture [12], new mobile phone specifications by Intel and NTT DoCoMo, targets the consolidation of two heterogeneous guest OSs (e.g., Linux and Windows Mobile): an operator guest OS for personal applications and an enterprise guest OS for business applications.

In terms of reliability, a hypervisor provides strong partitioning between virtual machines. The hypervisor places the guest OSs at a lower privilege level than itself and validates all the page table update requests of the guest OSs. This mechanism prevents one guest OS from reading from or writing on the system memory of the other guest OSs. Therefore, every virtual machine is isolated and protected from the others.

A hypervisor and guest OS on a CE device generally adopt a para-virtualization technique to minimize virtualization overhead than that of the full virtualization approach. The para-virtualization technique enables a collaboration between a hypervisor and a guest OS by providing communication channels named hyper calls, which are analogous to system calls between an operating system and a user application. The hyper calls request the hypervisor to execute the instructions on behalf of the guest OSs. Hyper calls include processor state update operations, for example, memory management unit (MMU) update and physical interrupt masking operations, that a guest OS cannot execute directly. Source codes of a guest OS should be modified in the para-virtualized approach to contain hyper calls.

2.2. Related Work

Xen on ARM is migrated from the open source Xen [7] that targets server consolidation. In contrast to the server Xen, Xen on ARM targets the consolidation of embedded OSs and the security features between the OSs. It adopts a para-virtualization technique that requires operating system modification.

Xen on ARM provides ARM CPU, memory, and I/O device virtualization. In CPU virtualization, it replaces sensitive instructions, which modify system states and are executed

in the supervisor mode, of guest OSs to hyper calls. In memory virtualization, it provides memory protection between a hypervisor, guest OSs, and applications by exploiting a domain protection mechanism as the ARM architecture has fewer privilege rings (two rings) than the x86 architecture. In device virtualization, domain0, which is an administrator virtual machine, runs most of the native device drivers; all the hardware interrupts are delivered to domain0 before they are redirected to the target virtual machines as virtual interrupts. This architecture degrades network performance because network packets of the target virtual machine are sent or received through domain0 [7].

Our multi-core hypervisor is extended from the single-core MobiVMM, which is designed for a soft real-time research of smart mobile phones. It is based on the TI OMAP2430 development platform; it implements ARM CPU virtualization, memory virtualization, and partial I/O virtualization, which is called pseudo I/O virtualization [15]. MobiVMM currently runs two Linux OSs as its guest OSs: guest OS1 for a soft real-time OS and guest OS2 for a general-purpose OS. Fig. 2 illustrates the structure of the single-core MobiVMM.

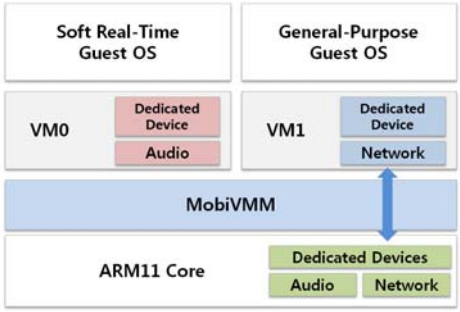

Fig. 2. Structure of the single-core MobiVMM

The MobiVMM design primarily considers the soft real-time property. MobiVMM uses a preemptive scheduler so that a soft real-time guest OS can get the processor time whenever it wants. We also provide low-latency I/O processing that cooperates with the scheduler to deliver interrupts and events to a soft real-time guest OS on time.

For I/O virtualization, we categorize devices into four groups on the basis of the usage type: dedicated, active, running, and dynamic devices. Dedicated devices are subject to a target virtual machine and are not shared among virtual machines. Active devices are shared devices and are active only when a user explicitly gives a control (e.g., key pad). Running devices are under control of the currently running virtual machine (e.g., hardware timer). Dynamic devices are fully shared devices.

VirtualLogix [11] is a commercial hypervisor for ARM-based processors, including ARM MPCore, but we are not aware of its technical details.

3. Hypervisor Design

In this section, we cover the details of the multi-core hypervisor design. In particular, we provide the virtualization methods for a multi-core processor, a physical memory, and I/O devices.

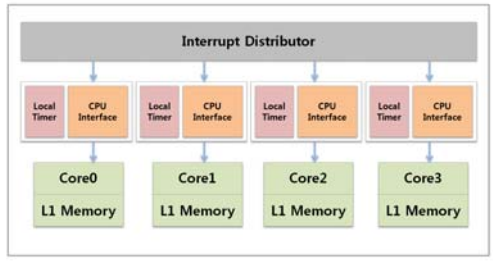

Fig. 3. Components of the ARM11 MPCore processor

3.1. Multi-core Processor Virtualization

The ARM11 MPCore processor consists of four major parts: four cores, local timers, CPU interfaces, and an interrupt distributor [9]. In

particular, each individual core implements ARM architecture v6 and has an L1 cache memory. For each core, each local timer generates timer interrupts. The CPU interfaces play the role of a communication channel between the cores and an interrupt distributor. They process interrupt acknowledgment, interrupt masking, and end of interrupt (EOI) acknowledgment. The interrupt distributor distributes hardware interrupts from physical devices to target cores. It also delivers inter-processor interrupts (IPIs) between cores. Fig. 3 illustrates all components of the processor.

For the virtualization of each individual core, the hypervisor replaces all of the sensitive instructions to hyper calls. The local timers, CPU interfaces, and interrupt distributor are also virtualized by the emulation of related instructions.

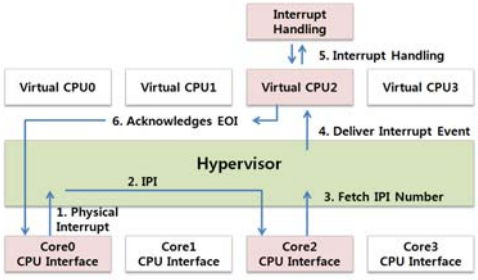

Fig. 4. Dynamic distribution of the virtual interrupt distributor

We extend the functionality of the virtualized interrupt distributor to scatter interrupts dynamically among cores. Originally, the ARM11 MPCore interrupt distributor does not support the dynamic distribution of interrupts [9], whereas the local APIC of x86 platforms provides both static and dynamic distribution [10]. In the dynamic distribution manner, interrupts are distributed in a round-robin fashion among cores by an arbitration mechanism. As the MPCore processor does not support dynamic distribution, all hardware interrupts are delivered to the first core (core0) in default settings. Therefore, only the first core is mainly utilized even though interrupt loads are heavy. In order to overcome this limitation, the virtual interrupt distributor code that runs on core0 distributes interrupts to other cores dynamically when core0 is heavily utilized. This functionality helps interrupt processing, which is composed of the top-half and the bottom-half (soft IRQ in Linux) processes, to be distributed automatically among all cores.

The dynamic distribution of the virtual interrupt distributor operates as illustrated in Fig. 4. When a hardware interrupt occurs at the first core, the hypervisor generates an inter-processor interrupt to the target core. Then, the target core recognizes what the interrupt is by fetching the IPI number and delivers an interrupt event to the virtual CPU on the target core. Then, the guest OS code on the target core executes the interrupt handling processes. After completing the processes, it acknowledges the end of an interrupt to the source core through the CPU interface alias of the source core as the interrupt occurs on the source core.

3.2. Memory Virtualization

The hypervisor shares the same address space with a guest OS in order to avoid frequent cache and TLB flush. It resides in the highest 32 MB of the 4-GB address space. Because the memory is an independent part not related to multi-core platforms, the implementation of the memory virtualization completely follows that of the single-core MobiVMM.

3.3. I/O Virtualization

Compared to Xen on ARM in which domain0 has all the native drivers, our hypervisor is designed to allow each guest OS to have native drivers. Each native driver is dedicated to a guest OS or shared between domains by

an I/O ring buffer mechanism that delivers the I/O requests and responses. This architecture helps to decrease the response time of a soft real-time virtual machine by preventing indirect access to native drivers. In our approach, each interrupt of the peripheral devices is delivered to the virtual machine that has the native device driver for that interrupt.

4. Subsystems in Hypervisor

In this section, we describe multi-core-related subsystems that manage and support virtual machines in the hypervisor.

4.1. Virtual Machine Scheduler

We have implemented a simple credit-based scheduler [13] as our default virtual machine scheduler because in the multi-core environment, it exhibits high performance and CPU fairness between virtual machines. In the credit-based scheduler, credit refers to the CPU time or CPU bandwidth for which each virtual machine can run. A virtual machine is weighted by the credit amounts, and if two virtual machines have the same amount of credits, the virtual machines can get an equal amount of CPU time.

Our credit scheduler provides a variable quantum (10 ms-50 ms) for a virtual CPU, according to the characteristics of the system. In order to improve responsiveness, a short quantum value (10 ms) is appropriate. A virtual CPU of a domain consumes its credit at every 10-ms time tick when it runs on a physical CPU. A virtual CPU with a positive credit value can be scheduled, and credits are redistributed to every virtual CPU periodically.

In addition to this principle, in order to guarantee a soft real-time property as that of the single-core MobiVMM, our credit scheduler supports preemptiveness.

When multimedia-related interrupts (e.g., audio interrupts) occur at one physical core, we immediately deliver the interrupts to the soft real time virtual machine by prioritizing interrupt handling. Then, we preempt the general-purpose virtual machine running on the core so that the soft real-time virtual machine can obtain processor time immediately.

4.2. Virtual and Physical Time Manager

The virtual and physical time manager supports both virtual and physical time for guest OSs by the virtualization of each local timer in the processor. Virtual time advances when the guest OS receives timer events from the hypervisor while the virtual CPU of each guest OS is running. The virtual time is used for scheduling of processes in a guest OS. The physical time advances irrespective of whether each guest OS is running or not. A guest OS can use physical time whenever it needs to know the time passed since the boot of the system. For this purpose, the manager maintains a 64-bit global jiffy variable per local timer, which increases by 1 when the local timer generates a timer interrupt. The hypervisor exports the variable through a hyper call named vmm_get_global_jiffies() to each guest OS.

4.3. Startup Module: Starting a New Virtual Machine

The startup module in the hypervisor starts a new virtual machine. When a control application in the first virtual machine (VM0) notifies the start of a new virtual machine to the startup module through a hyper call, the startup module prepares an initial page table that is necessary for the boot process of the new virtual machine. After preparing the page table, the module signals core0 to execute the first kernel code of the new virtual machine (in Linux, /arch/arm/kernel/head.S). The new virtual machine skips the process of building an initial page table because the process assumes that the MMU is off. Instead, it uses the page table delivered from the hypervisor.

At first, core0 only participates in the boot process; after an early system boot, other cores are activated by IPIs from core0.

5. Network Performance Improvement

In this section, we elaborate on how network performance can be improved with our hypervisor in a CE device. As an explanation, we provide an example of a virtualized CE device using the network.

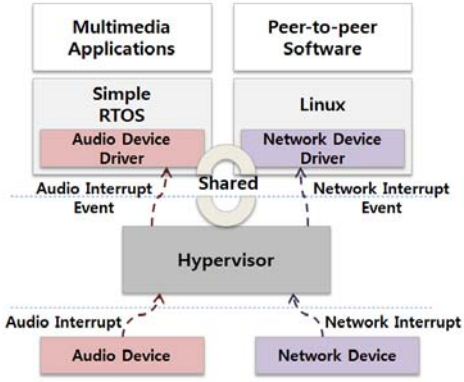

Fig. 5. Configuration of the virtualized CE device

5.1. Example of Virtualized CE Device Using Network

Fig. 5 describes the configuration of a virtualized CE device using the network. Practically, a virtualized CE device can be composed of a soft real-time virtual machine, which executes soft real-time applications such as MP3 and H.264 decoders, and a general-purpose virtual machine for Internet use. For example, a digital TV system can load a simple RTOS with an exclusive media player for TV and a Linux OS with an Internet browser or peer-to-peer (P2P) software.

As our hypervisor allows each guest OS to have native drivers that can be dedicated or shared, it is possible to place a native driver for an audio device into the simple RTOS and to place a network driver into the general-purpose OS for decreasing the response time. In this architecture, audio interrupts are delivered to the soft real-time virtual machine; network interrupts are delivered to the general-purpose virtual machine.

5.2. Network Performance Problem

When we use the abovementioned device with no support of dynamic distribution of interrupts, core0 spends most of the CPU bandwidth on the soft real-time virtual machine. When a user simultaneously executes multimedia applications and peer-to-peer software at each virtual machine, both audio and network interrupts occur continuously on core0. In order to play the sound of multimedia applications smoothly, we immediately deliver audio interrupts to the soft real-time virtual machine as interrupt events and preempt the general-purpose virtual machine if it is running. However, the problem is that the audio interrupts occur periodically and frequently, i.e., they occur 3-4 times per 10 ms time tick; for reference, network interrupts mostly occur once per time tick in a single run of a network program. Even if the general-purpose virtual machine is selected to run by the scheduler, the virtual machine is preempted periodically within its time quantum by audio interrupts. Therefore, the soft real-time virtual machine acquires most of the CPU bandwidth of core0. Fig. 6 illustrates this situation.

Conversely, because the general purpose virtual machine receives CPU bandwidth less than from what it normally should, network performance suffers. The insufficient CPU bandwidth prevents the steady processing of network interrupt handling and causes delay in the occurrence of a next network interrupt.

We have measured the bandwidth of the

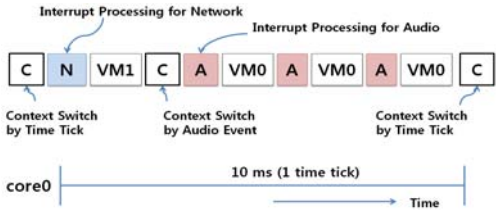

Fig. 6. Interrupt processing on core0. Note that we use a preemptive scheduler so that the soft real-time virtual machine can obtain processor time immediately when interrupts occur

network by using the Iperf benchmark program in the general-purpose virtual machine. When the soft real-time virtual machine is idle, the result is 15.7 Mb/s on an average. However, when the soft real-time virtual machine plays an mp3 file, the result is 2.30 Mb/s on an average.

5.3. Solution for Performance Problem

In order to improve network performance, the hypervisor scatters the workloads to other cores that are idle or underutilized. This process is executed by the help of the virtual interrupt distributor and the hypervisor profiler. When both audio and network interrupts occur on a single core concurrently, the virtual interrupt distributor decides whether it will deliver one of the interrupts among other underutilized cores or not. The decision is based on the information given by the profiler in the hypervisor.

The profiler separately maintains the frequency of audio and network interrupts during some intervals (e.g., 100 ms). The profiler then analyzes whether each frequency of audio and network interrupts is periodic or sporadic in the intervals. If both frequencies are periodic, the profiler informs the virtual interrupt distributor of the underutilized core that will be a target of the distribution. Then, the distributor sends one of the interrupts to the above informed core. If the dynamic distribution is once started by the distributor,

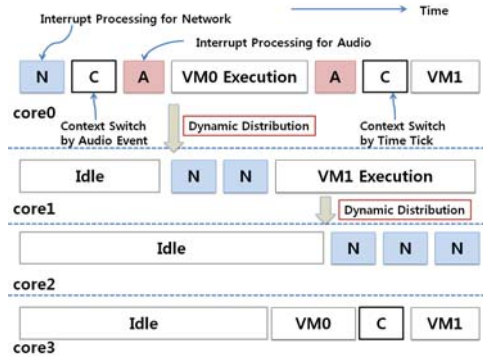

Fig. 7. Scattered interrupt processing among cores

the profiler continuously checks the CPU utilization of each core so as to guarantee the performance of interrupt processing. If any frequency becomes sporadic, the dynamic distribution is completed in order to prevent the inter-processor interrupts (IPIs) overhead caused by the processing of the interrupt distribution. Fig. 7 illustrates scattered workloads among core0, core1, and core2 when the network interrupt is the target of distribution.

6. Evaluation

We have implemented the hypervisor on an ARM11 MPCore platform that has four 250-MHz ARM11 cores. Fig. 8 shows our evaluation board with an ARM11 MPCore core tile. We use two para-virtualized guest OSs whose kernels are symmetric multiprocessors (SMP) Linux 2.6.21.

For experiments, we regard guest OS1 as the soft real-time OS and guest OS2 as the general-purpose OS. Using the Iperf network benchmark program, we have measured the network bandwidth in the general-purpose virtual machine.

6.1. Network performance without the virtual interrupt distributor

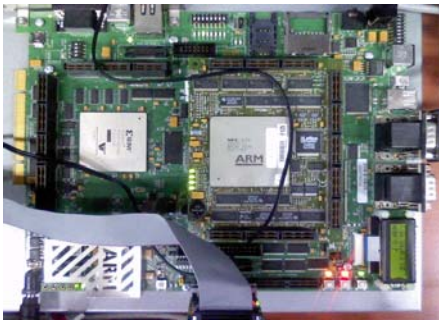

Fig. 8. Evaluation board with an ARM11 MPCore core tile

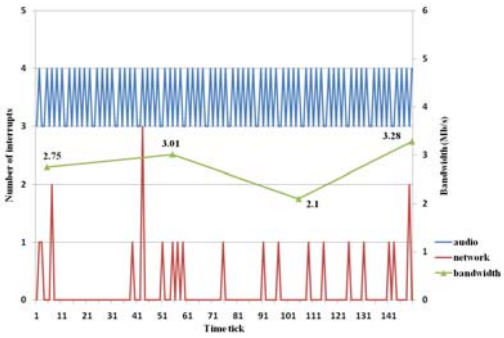

Fig. 9. Measurement of bandwidth and number of interrupts while the soft real-time virtual machine plays an mp3 file in the sampling interval (150 time ticks)

We have measured the network bandwidth while the soft real-time virtual machine is idle. The result is 15.7 Mb/s on an average. This result is a performance baseline. Then, while the soft real-time virtual machine plays an mp3 file, we have measured the network bandwidth together with the number of network and audio interrupts received by the hypervisor at every time tick (every 10 ms). Fig. 9 shows the result in a sampling interval (150 time ticks = 1.5s).

In this experiment, the network bandwidth is 2.30 Mb/s on an average and the network interrupts rarely occur. As the general-purpose virtual machine obtains a small amount of CPU bandwidth on core0, the processing of network interrupt handling is delayed. However, the number of audio interrupts (3~4 times per time tick) is the same as the number of interrupts in a single run of an audio program owing to the support for the soft real-time virtual machine.

6.2. Network performance with the virtual interrupt distributor

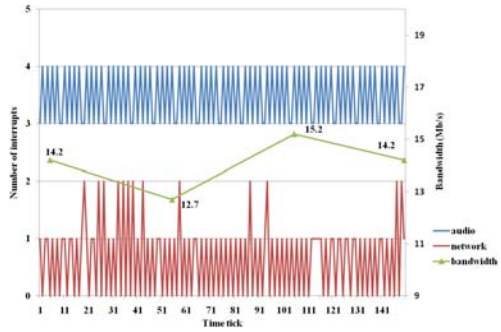

Fig. 10. Measurement of bandwidth and number of interrupts while the interrupt distributor works in the sampling interval (150 time ticks)

We have measured the network bandwidth and the number of interrupts while the virtual interrupt distributor works. Fig. 10 shows the result in a sampling interval (150 ticks = 1.5s). In this experiment, the network interrupt is redirected to core1 with the help of the virtual interrupt distributor and the profiler. As the distribution of network interrupts enables core1 to participate in interrupt processing, the frequency of network interrupts is rapidly increased when compared with the experiment in Fig. 9. Therefore, the network bandwidth achieves 13.4 Mb/s on an average. The difference between the performance baseline (15.7 Mb/s) and this result is due to the inter-processor interrupts (IPIs) overhead between core0 and core1, which is illustrated in Fig. 4.

6.3. Other cases

We have also measured the network bandwidth while the soft real-time virtual machine executes an H.264 decoder. The

H.264 media file is encoded for parallel decoding and the display resolution is 352 × 288. Because the media file does not contain audio data, we have used an mp3 player instead. In this experiment, without the virtual interrupt distributor, the network bandwidth is 2.21 Mb/s on an average. However, with the virtual interrupt distributor, the network bandwidth is 11.6 Mb/s on an average. Compared to the performance baseline (15.7 Mb/s), the network performance somewhat degrades because cores1-3 become busy decoding H.264 frames.

6.4. Summary

Table 1 summarizes all the benchmark results. Compared to case no. 2 and case no.5 whose bandwidths are in the range 2.21-2.30 Mb/s, our approaches (case nos. 3 and 4 and case nos. 6 and 7) improve the network performance up to 5.5 times.

Table 1. Iperf benchmark result

Case no.	Audio	Network	Bandwidth (Mb/s)
1. max bandwidth	no audio	core0	15.7
2. only core0	core0	core0	2.30
3. distribution #1	core0	core1	13.4
4. distribution #2	core0	core1-3	13.0
5. only core0 with H.264	core0	core0	2.21
6. distribution #3 with H.264	core0	core1	11.6
7. distribution #4 with H.264	core0	core1-3	11.5

7. Conclusion

In this paper, we presented a new hypervisor design and implementation for multi-core CE devices. With our hypervisor, the requirements for strong security, user extensibility, and acceptable performance of the CE device could be met with less effort.

In particular, we described the details of the hypervisor design and multi-core-related subsystems. Also, we elaborated on network performance improvement in a virtualized CE device. With the help of the new dynamic virtual interrupt distributor in the virtualization layer, our approach improved the network performance up to 5.5 times.

References

[1] J. Y. Hwang, S. B. Suh, S. K. Heo, C. J. Park, J. M. Ryu, S. Y. Park, and C. R. Kim, "Xen on ARM: System virtualization using Xen hypervisor for ARM-based secure mobile phones", Proc. CCNC, Jan. 2008.

[2] S. B. Suh, "Secure architecture and implementation of Xen on ARM for mobile devices", Proc. 4th Xen Summit, 2007.

[3] G. Heiser, "Hypervisors for consumer electronics", Proc. CCNC, Jan. 2009.

[4] H. Kondo, O. Yamamoto, S. Otani, N. Sugai, and T. Shimizu, "Software architecture of a secure multimedia system using a multicore SoC and software virtualization", Proc. ICCE, Jan. 2009.

[5] Z. Tan, S. Zheng, J. Hu, Y. Chen, and P. Liu, "Design and implementation of the software system on MPSoC: An HDTV decoder case study", IEEE Trans. Consumer Electron., Vol.52, No.4, pp. 1333-1339, Nov. 2006.

[6] T. R. Jacobs, V. A. Chouliaras, and D. J. Mulvaney, "Thread-parallel MPEG-2, MPEG-4 and H.264 video encoders for SoC multi-processor architectures", IEEE Trans. Consumer Electron., Vol.52, No.1, pp. 269-275, Feb. 2006.

[7] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization", Proc. SOSP, Oct. 2003.

[8] J. Goodacre and A. N. Sloss, "Parallelism and the ARM instruction set architecture", Computer, Vol.38, No.7, pp. 42-50, July. 2005.

[9] ARM. ARM MPCcore Processor - Technical Reference Manual, Revision: r1p0.

[10] Intel corp. IA-32 Intel Architecture Software

Developers' Manual: System Programming Guide, Aug. 2007. http://developer.intel.com/.
[11] VirtualLogix, Real-Time Virtualization, http://www.virtuallogix.com/.
[12] Open and secure terminal initiative (OSTI) architecture specification. http://www.nttdocomo.co.jp/english/corporate/technology/osti/, Oct. 2006.
[13] L. Cherkasova, D. Gupta, and A. Vahdat, "Comparison of the three CPU schedulers in xen", ACM SIGMETRICS Performance Evaluation Review, Vol.35, No.2, pp. 42-51, Sep. 2007.
[14] Z. Tan, S. Zheng, P. Liu, G. Lin, and S. Yu, "An implementation of open source operating system on multiprocessor system-on-a-chip", IEEE Trans. Consumer Electron., Vol.52, No.3, pp. 1118-1123, Aug. 2006.
[15] S. Yoo, Y. Liu, C.-H. Hong, C. Yoo, and Y. Zhang, "MobiVMM: a virtual machine monitor for mobile phones", Proc. MobiVirt, June. 2008.
[16] S. Yoo, M. Park, and C. Yoo, "A step to support real-time in virtual machine", Proc. CCNC, Jan. 2009.

BIOGRAPHY

Cheol-Ho Hong

received the B.S. and M.S. degrees in computer science and engineering from Korea University, Seoul, Korea, in 2001 and 2003, respectively.

He worked as a senior researcher at Netville, Seoul, Korea, from 2003 to 2006. Currently, he is a Ph.D. candidate of Korea University, Seoul, Korea. His research interests include hypervisor, multi-core architecture, and embedded system.
Email : chhong@os.korea.ac.kr

Miri Park

received the B.S. and M.S. degree in computer science and engineering from Korea University, Seoul, Korea, in 2006 and 2010, respectively.

She worked as an engineer at Motorola, Seoul, Korea, from 2006 to 2007. Currently, she is an engineer at LG Electronics, Seoul, Korea. Her research interest is the system virtualization for RTOS.
Email : miri.park@lge.com

Seehwan Yoo

received the B.S. and M.S. degrees in computer science and engineering from Korea University, Seoul, Korea, in 2002 and 2004, respectively.

He is currently a Ph.D. candidate at Korea University, Seoul, Korea. His current interest is system virtualization.
Email : shyoo@os.korea.ac.kr

Chuck Yoo

received the B.S. degree in electronic engineering from Seoul National University, Seoul, Korea and the M.S. and Ph.D. in computer science from University of Michigan.

He worked as a researcher in Sun Microsystems Laboratory, from 1990 to 1995. He is now a professor in department of computer science and engineering, Korea University, Seoul, Korea. His research interests include high-performance networks, multimedia streaming, and operating systems. He served as a member of the organizing committee for NOSSDAV 2001.
Email : hxy@os.korea.ac.kr