Optimizing the Startup Time of Embedded Systems: A Case Study of Digital TV

Heeseung Jo, Hwanju Kim, Jinkyu Jeong, Joonwon Lee, and Seungryoul Maeng

Abstract — *The demand for fast startup is mostly motivated by embedded systems, especially for home appliances such as digital TV. Though a new storage device such as a flash memory may help reduce the startup time, its applicability is limited to a small size application, which is not the case for recent media processing software consisting of millions lines of code. This paper proposes novel approaches to reduce the startup time of large embedded systems. The startup latency of a commercial digital TV is analyzed in order to marshal resource initialization and to warm up the buffer cache. Based on the analysis we propose a better initialization order and determine when data should be prefetched to the buffer cache in order to reduce the startup time, and which data. Our measurements show that our schemes reduce the total startup time by 35%*[1].

Index Terms — Booting time, Startup time, Embedded system, Digital television (DTV)

I. INTRODUCTION

With the evolution of VLSI technology, the computing power of embedded systems has recently been enhanced significantly, and thus they can accommodate software with rich functionality. Many embedded systems deal with multimedia, which requires complex and huge software along with many esoteric software libraries. We believe that this trend will continue for the foreseeable future, since the standards for media processing are becoming more complex, and manufacturers want to satisfy diverse user requirements with a single device [9], [10].

A digital TV (DTV) is a representative case of this complexity, because a system must support different broadcasting standards and various media formats in order to compete in the global market. Furthermore, the behavior of a TV is expected to be similar to other home appliances which rarely crash. This high reliability also increases the size and complexity of the software dramatically. Other embedded systems such as personal media players, navigators, and cellular phones show similar trends of hosting large software due to the digital convergence trend.

The demand for fast startup is mostly motivated by embedded systems, because it is one of the most important factors affecting the decisions of buyers. A long waiting time will alienate impatient users. The convergence of functionalities puts many devices onto a single system, which results in a long startup for their initialization. Using large software to boot up, this latency severely prolongs the startup time of an embedded system.

To analyze and reduce the startup latency, a commercial digital TV was selected for the case study, since it is a representative of many embedded systems and has the highest software complexity. Aside from traditional TV reception, recent digital HDTV (High Definition TV) functions as a computer providing internet access, a photo album library, and a game station, all with high quality images. Complex standards of terrestrial and cable digital TV broadcasting also contribute to the high complexity of the system.

For current commercial digital TVs, it takes 6-15 seconds for the first screen to appear. This latency comprises OS booting, initializing hundreds of hardware registers, and launching applications. Optimizing the boot up latency of the OS has been studied in many areas. The register initialization process is difficult to optimize, because of the dependencies among register values and the features unknown to software developers. Most of the time consumed for launching applications involves reading codes from hard disk or flash memory.

Based on our measurements, we found that the bottlenecks are usually CPU or IO devices, depending on which actions are performed. But overall utilizations of both components are low, meaning the current startup process needs to be optimized in order to enhance the utilizations. Based on these measurements, we propose two approaches to reduce the startup time by making the system maximize its hardware utilization. Our approaches do not require any modification to existing software including the kernel. The proposed mechanisms are useful especially for embedded systems with complex hardware and large applications.

II. DTV SYSTEM

A. DTV Architecture

The DTV system used in this work is a current commercial product. Its hardware is composed of an

[1] This work was supported by the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea government (MEST) (No. 2009-0080381) and the MKE (Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) Support program supervised by the IITA (Institute of Information Technology Advancement) (IITA-2009-C1090-0902-0020).

Heeseung Jo, Hwanju Kim, Jinkyu Jeong, and Seungryoul Maeng are with the Korea Advanced Institute of Science and Technology (KAIST) (e-mail: heesn@camars.kaist.ac.kr, hjukim@camars.kaist.ac.kr, jinkyu@camars.kaist.ac.kr, maeng@camars.kaist.ac.kr).

Joonwon Lee (corresponding author) is with the Sungkyunkwan University (e-mail: joonwon@skku.edu).

Contributed Paper
Manuscript received October 7, 2009

embedded CPU [2], 128 MB memory, and NAND flash memory as a storage device. Additionally, there are several special devices designed for DTV. Especially, the tuner and the decoder devices are important for DTV functionality. The tuner device receives a video stream from a broadcasting station, and the decoder device is responsible for decoding the stream. Besides, graphic and sound devices are also needed.

Figure 1 shows the software architecture of the DTV system used in this work. It consists of a bootloader, an OS, device drivers, and applications. This system uses an embedded bootloader [3] and an embedded OS. The booting sequence is the same as that of other embedded systems. After loading the kernel, the system initiates a kernel module named *settop* that manages DTV-specific devices. This module is a thin layer which is used as a medium to connect user-level applications with the kernel.

The DTV system in our work utilizes an integrated library, called *libsettop*, which controls the overall DTV devices in the user-level mode. The library provides a group of APIs for the application software. Using this library, the DTV application initializes the hardware devices and displays the decoded video stream, menu, and channel information on the screen.

The DTV application named *exeDTV* is a multi-threaded application that provides the general DTV functionalities. In addition to the basic DTV functionalities, it manages menu navigation, the control of video and sound qualities, and other non-television services. It is a huge user-level application that consists of 3.4 million lines of C++ code, and its binary image size is about 8 MB. In the beginning, it initializes all necessary devices and reads resource files such as menu images, fonts, etc. These resource files are about 134 MB, and they are stored in NAND flash storage separately from the exeDTV binary image [8].

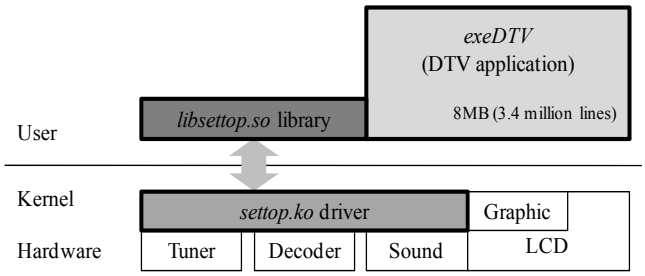


Fig. 1. The software architecture of the DTV system.

B. Normal Startup Time and Resource Consumption Analysis

In this paper, we define a startup as *completed* when the first display appears on the screen, since common users perceive screen-on as the power-on of a television set. Note that the startup time in this paper denotes the elapsed time from the cold power-down state to the DTV screen-on state. Therefore, the startup sequence must pass through the entire software stack including the bootloader, the kernel, device drivers, and the DTV application, to be completed.

The normal startup analysis of DTV is shown in Figure 2. The x-axis of the figure is the execution time in seconds after DTV power-on. The upper graph shows the CPU usage and the amount of read I/O. The CPU usage is classified into idle, system, and user. The lower graph shows the consumed time for each section of the startup procedure. We basically divide the startup time into three parts: the kernel, the init script, and the DTV application. In this analysis, the execution time of the bootloader is excluded, since it is negligible and almost constant. Moreover, the resource usage of the boot loader cannot be measured exactly.

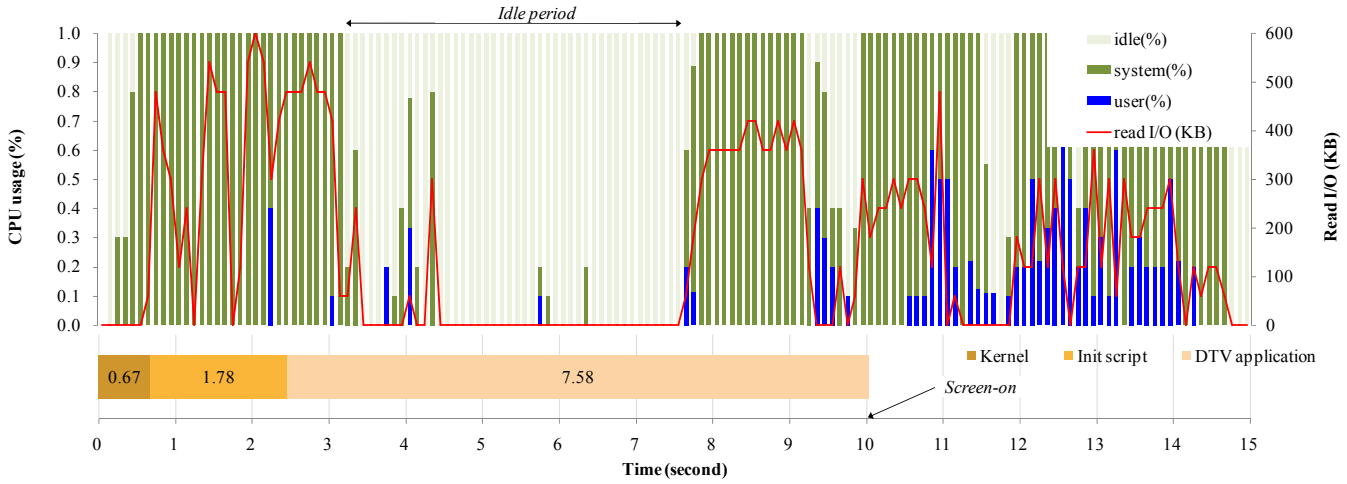


Fig. 2. The startup time and resource utilization of the DTV system

TABLE I
THE NORMAL SCRIPT (*RC.LOCAL*) AND THE MODIFIED SCRIPTS (*RC.TOP-HALF* AND *RC.BOTTOM-HALF*)

rc.local
1. Flash memory driver load
2. USB driver load
3. ***Settop driver load ****
4. ***Mount file systems (1) ****
5. Mount file systems (2)
6. ***Launch the DTV application ****

rc.top-half
1. Settop driver load
2. Mount file systems (1)
3. rc.bottom-half &
4. Launch the DTV application

rc.bottom-half
1. Wait 3 seconds
2. Flash memory driver load
3. USB driver load
4. Mount file sytems (2)

The kernel booting takes 0.67 seconds, as its CPU usage is relatively high. This kernel is an optimized version of fast bootup from the vanilla kernel based on previous work [4]-[7]. The init script is in charge of mounting file systems, loading modules, and launching the DTV application. It takes about 1.78 seconds, and fully utilizes CPU resources. The large time consumption for the read I/O operations of the init script involves mounting the file system and loading the images of the DTV application. The DTV application part is the elapsed time between the start of the DTV application and the screen-on, and it is about 7.58 seconds. It is notable that during the early stage (3-8 seconds) the DTV application is almost idle. Based on a more detailed analysis, we found that the idle period is a result of initializing the DTV-specific devices. The overall time consumption for the startup of a DTV system is 10.03 seconds.

III. REDUCING THE STARTUP TIME

This section describes our two approaches to reduce the startup time of our DTV system. As shown in the previous section, the system utilization is low when the DTV application is idle during the early stage. During this period, hardware devices are initialized, and this process cannot be accelerated, since it is sequential. Therefore other startup tasks that can be executed in parallel during this period must be found in order to increase the resource utilization.

We exclude approaches modifying existing hardware or software such as the kernel, modules, libraries, or the DTV application. Such modifications require a great deal of engineering when the software system is changed, which is done each time a new hardware platform is introduced. Most manufacturers deploy several platforms at once in order to be competitive in different market sectors, and platform changes are common.

A. Reordering resource initialization

To determine the parallelism in the startup process, especially in the initialization process, each initialization script is analyzed. The most time-consuming task needs to be started at the earliest possible time in order to be overlapped with other tasks and to enhance resource utilization.

The *init* script of OS, *rc.local*, manages triggering the initialization of hardware devices and launching the DTV application. Table I (a) shows the original rc.local script sequence, and (b) shows the modified one. The rc.local script loads the needed device driver modules first, and then mounts the file systems. In our DTV system, the NAND flash memory used as storage is divided into several partitions for various purposes. Most partitions are read-only, but two are read-write partitions to store user configuration and channel information.

In the normal script, the lines marked with asterisks should be completed before launching the DTV application with the time-consuming hardware initialization process. To start the DTV application at the earliest possible time, all these lines of operations should be executed as fast as possible. The dependencies among those prerequisite operations are honored, and they are collected into the *rc.top-half* script. The remaining operations are executed from the *rc.bottom-half* script, which is run in the background.

Once the rc.bottom-half script starts, it runs concurrently with the DTV application in order to boost the low resource utilization measured during the idle period of the DTV application. The issue here is that the DTV application is not idle in the beginning. Since our goal is to finish the DTV application as soon as possible, while sharing resources with the operations defined in the rc.bottom-half script, the rc.bottom-half script waits until the DTV application arrives at the idle period. Based on several experiments we found that the optimal waiting time is three seconds. The DTV application does not have to wait until rc.bottom-half finishes the most time-consuming tasks such as mounting file systems and loading kernel modules, because these are accessed later.

B. I/O interleaving

I/O operations are far more time-consuming than the operations performed on the CPU or the main memory. For the DTV case, most I/O operations during the startup are flash memory accesses. Programs and data are read into the main memory by the file system. If a paging is used, page faults put new pages into the main memory.

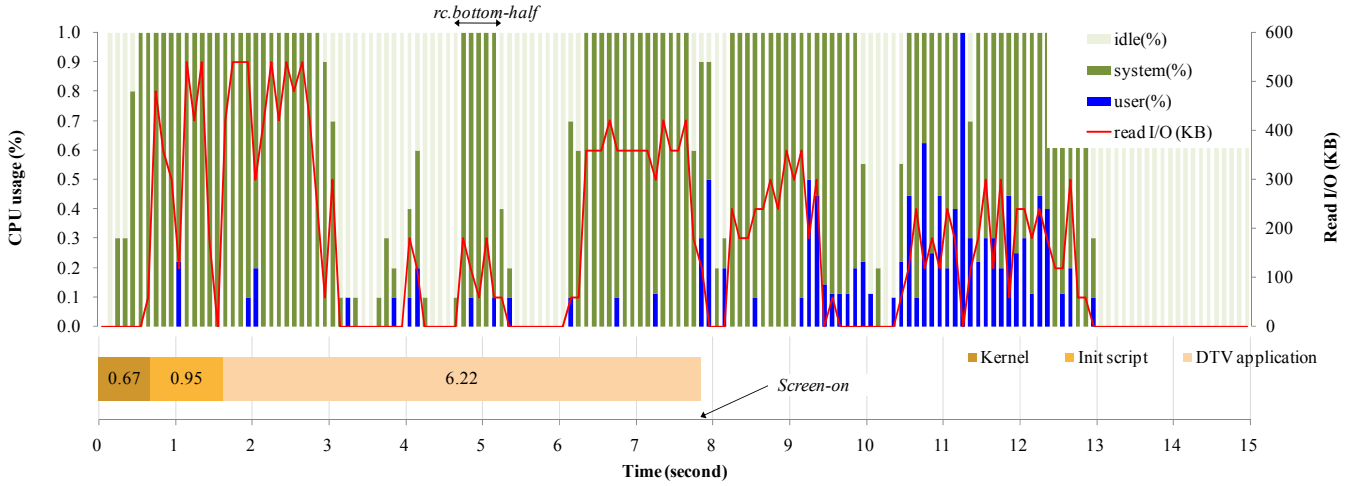


Fig. 3. The startup time and resource utilization of the DTV system using script relocation

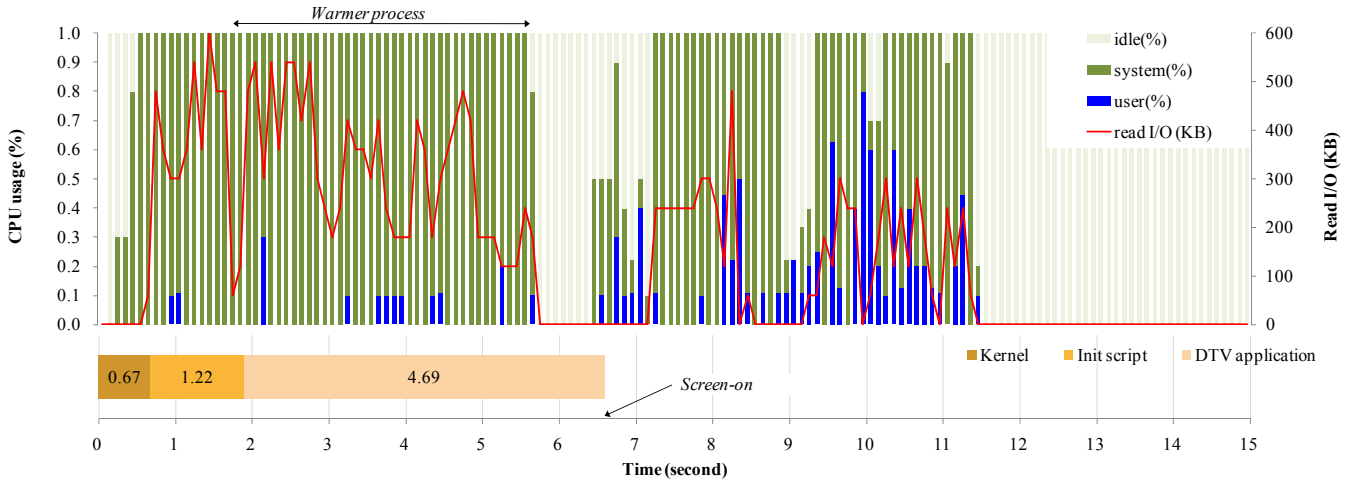


Fig. 4. The startup time and resource utilization of the DTV system using I/O interleaving with script relocation

A buffer cache is devised to cache recently used data in the memory. If a request to the flash memory can be serviced by the buffer cache, the latency can be greatly reduced. During the startup period, a great deal of data is read from the flash memory, but Figure 2 shows that intensive I/O operations are overlapped with high CPU utilization periods, which means that I/O operations may delay the startup process. If they can be executed during the idle period, the unnecessary delay of the CPU intensive tasks can be avoided.

For ease of programming and protection between applications, most systems employ demand paging, which puts into the memory only the pages needed for the current computation. When a page access is issued, a page fault is raised if it does not exist in the memory. A fault is an interrupt, and thus it traps into the kernel. The fault handler inside the kernel puts the requested page into the memory. This process is very time-consuming for processing the interrupt and for reading the page from the flash memory. Page faults can be reduced by preloading the pages needed for the start up process.

I/O interleaving is an approach to improve the DTV startup time by warming the buffer cache in advance and reducing major page faults. The set of programs and data that are needed during the startup procedure is deterministic, and the order of accesses is fixed. Using this information, I/O operations can be spread over the idle period in order to enhance resource utilization. To utilize the initial idle time of the DTV application, we made a lightweight task, which reads the pages of the DTV application and the needed resources in advance. It is initiated before the DTV application is launched, and runs concurrently as a background process. During the initial idle time of the DTV application, the process warms the buffer cache for the DTV application, thus, the DTV

application can avoid some of the page faults. The thread is destroyed as soon as it finishes reading the specified files. Consequently we can utilize the system idle period by letting the warming process fill the buffer cache.

This approach raises several issues. First, the files to be read by the warming thread should be decided. We profiled the files that are required by the DTV application until screen-on, and these were filtered according to the requested order and file size, which determines the number of page faults. Another issue is the case where warming does not finish its job within the idle time. The unread files, however, are read by the DTV application or warming after all. This does not adversely affect the startup time, while the benefit from warming is large. The last issue is whether the thread should run in the user mode or in the kernel mode. We evaluated both and selected the user-level process based on the engineering cost, since the startup times were almost equal.

IV. EVALUATION

Figure 3 shows the startup time of the DTV system using script relocation. The startup time of the kernel is equal to that of the normal kernel. The init script takes 0.95 seconds, which is the time consumed by rc.top-half. This is almost half of the normal init script. The rc.bottom-half is executed with the DTV application concurrently. The rc.bottom-half is initiated at about the fifth second, which is in the middle of the idle time. Eventually, in our script relocation approach, the execution time of rc.bottom-half is absorbed into the idle time, thereby advancing the DTV application launching time by 0.83 seconds. The execution of the DTV application until screen-on takes 6.22 seconds, and this is reduced by 1.37 seconds, since the earlier start time of the DTV application causes a faster screen-on. The CPU usage graph shows that the idle time is reduced, and the total startup time is 7.84 seconds, which is a 22% reduction of the normal startup time.

Figure 4 shows the startup time of the DTV system using I/O interleaving with script relocation. Although the time of the init script is slightly increased to launch warming, the DTV application execution time is significantly reduced to 4.89 seconds. This remarkable reduction of the startup time is mainly due to the reduced page faults in the initial idle time of the DTV application. In the CPU usage graph, the idle time is eliminated, since it is consumed by warming. The overall startup time is 6.58 seconds, which is about a 35% reduction of the normal startup time.

V. RELATED WORK

There has been a lot of previous work to improve the startup time of embedded systems. Bird proposed several techniques to enhance the booting time in a commodity OS [4]. He analyzed each step of the boot sequence and suggested various methods such as kernel execution-in-place (XIP), probe delay elimination, RTC read synchronization, and so forth. Wool proposed methods to optimize the boot time for an embedded OS [7]. His approach not only concerns the time reduction needed to initialize the core kernel and drivers but also addresses application-level considerations and other areas of the kernel such as linear flash memory access and time-optimized module insertion. Park et al. applied the well-known boot time reduction techniques to a commercial product [6]. They targeted a digital camera for their research. Through their experiments, they analyzed the detailed boot time consumption for each boot step.

Kaminaga proposed a snapshot boot mechanism to enhance the startup time of a commodity OS by using the suspend-resume technique [5]. The snapshot boot technique suspends the running kernel, and resumes with the suspend image instead of booting. To reduce resume time, it resumes the suspend image from a boot loader, not the kernel. With the snapshot boot technique, he also applied other techniques such as XIP, pre-linking. Jo et al. adopted the snapshot boot mechanism into a DTV system [8]. They included the DTV application into the snapshot image to enhance the startup time of the application. Especially, they proposed two different approaches that can be customized between the startup time and the engineering cost.

VI. CONCLUSION

As the computing power of embedded systems grows, their applications are becoming larger and more complex. In this paper, we proposed approaches to improve the startup time of these systems by increasing the utilization of system resources during the startup procedure. Especially, based on a case study on a commercial DTV system, we analyze its startup time in detail and evaluate our two approaches on the DTV system.

Based on the detailed analysis, we found that a large fraction of the startup time is taken for device initialization, and this can be reduced by utilizing the idle period. Considering the importance of time-to-market for embedded systems, our approaches can be adopted in existing systems without any modification to the kernel and its application. Our approaches are practical methods to enhance the startup time of a large embedded system other than a digital TV.

REFERENCES

[1] Y. Wu, S. J. Hirakawa, and U. H. Reimers, "Overview of Digital Television Development Worldwide," In *Proceedings of The IEEE*, vol.94, no.1, pp.8-21, Jan. 2006.

[2] D. Seal, "ARM Architecture Reference Manual, 2nd edition," *Addison-Wesley Longman Publishing Co., Inc*.

[3] C. Hallinan, "Embedded Linux Primer: A Practical, Real-World Approach," *Prentice Hall*, 2006

[4] Tim R. Bird, "Methods to Improve Bootup Time in Linux, Sony Electronics," In *Proceedings of the Linux Symposium*, 2004.

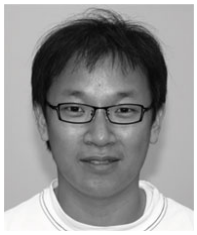
[5] Hiroki Kaminaga, "Improving Linux Startup Time Using Software Resume," In *Proceedings of the Linux Symposium*, 2006.

[6] Chanju Park, Kyuhyung Kim, Youngjun Jang, and Kyungju Hyun, "Linux Bootup Time Reduction for Digital Still Camera," In *Proceedings of the Linux Symposium*, 2006.

[7] Vitaly Wool, "Optimizing boot time for Embedded Systems," In *Proceedings of Free and Open Source Software Developer's European Meeting (FOSDEM)*, 2006.
[8] Heeseung Jo, Hwanju Kim, Hyun-Gul Roh, and Joonwon Lee, "Improving the Startup Time of Digital TV," *IEEE Transactions on Consumer Electronics,* Vol. 52, No. 2, pp. 485-493, May 2009.
[9] Ling Xing, Jianguo Ma, Xian-He Sun, Youping Li, "Dual-Mode Transmission Networks for DTV," *IEEE Transactions on Consumer Electronics*, Vol. 54, No. 2, May 2008.
[10] Charles W. Rhodes, "Interference to DTV Reception due to Non-Linearity of Receiver Front-Ends," *IEEE Transactions on Consumer Electronics*, Vol. 54, No. 1, Feb. 2008

Heeseung Jo received the B.S. degree in computer science from Sogang University, Korea, in 2000, and worked as an engineer at the mobile platform service team of KTF, Korea, from 2001 to 2004. He received the M.S. degree in computer science from Korea Advanced Institute of Science and Technology (KAIST) and is a Ph.D. candidate of KAIST. His research interests include virtual machine, flash memory, storage system, and embedded system.

Hwanju Kim received his B.S. degree in information and computer engineering from Ajou University, Korea, in 2006, and M.S. degree in computer science from Korea Advanced Institute of Science and Technology (KAIST) in 2008. He is a Ph.D. candidate of KAIST, and his research interests include virtual machine, distributed system, and storage system.



Jinkyu Jeong received the B.S. degree from the Computer Science Department, Yonsei University, and the MS degree in computer science from the Korea Institute of Science and Technology. He is currently a Ph.D. candidate in the Computer Science Department, Korea Advanced Institute of Science and Technology. His current research interests include real-time system, operating system, virtualization, and embedded system.

Joonwon Lee received the B.S. degree from Seoul National University in 1983 and the Ph.D. degree from the Georgia Institute of Technology in 1991. After working for IBM, he joined the faculty of the Korea Advanced Institute of Science and Technology (KAIST) in 1992. Currently, he is the faculty of the Sungkyunkwan University. His research interests include operating systems, low-power computing, and virtual machine.

Seungryoul Maeng received the B.S. degree from Seoul National University in 1977 and the M.S. and Ph.D. degree from the Korea Advanced Institute of Science and Technology (KAIST) in 1979 and 1984, respectively. He joined the faculty of the KAIST in 1984. His research interests include CPU architecture, parallel processing, cluster computing, and embedded systems.