Virtual-ROM: A New Demand Paging Component for RTOS and NAND Flash Memory Based Mobile Devices

Hyojun Kim, Jihyun In, DongHoon Ham,
SongHo Yoon, and Dongkun Shin

Samsung Electronics, Software Laboratories, Mobile Software Platform Team
416 Maetan-3Dong, Yeongtong-Gu, Suwon-City, Kyenggi-Do, Korea 443-742
{zartoven, jh_in, dh.ham, bluesol, dongkun.shin}@samsung.com

Abstract. Similiar to a hard disk, NAND flash memory must be accessed in sector unit, and cannot be used for code storage without copying its contents to RAM. A virtual memory technique is promising as a RAM saving solution. However, it can not be easily used without the operating system supports, and it is not suitable for real time systems because it causes unpredicted execution delays.

Virtual-ROM is a light-weight demand paging solution designed for RTOS based mobile devices. It is OS-independent, easy-to-use, and well optimized for NAND flash memory. Because it occupies only a restricted address space, real time tasks can be free from unpredictable execution delays by being excluded from Virtual-ROM. Our trace driven simulation showed that its performance is similar to 70ns NOR flash memory, and our real taget adaptation for a CDMA mobile phone showed that it saved about 30% RAM usage.

Keywords: Mobile, Virtual Memory, Demand Paging, Shadowing, RTOS, NAND Flash Memory.

1 Introduction

Flash memory is widely used for mobile devices due to its versatile features such as non-volatility, solid-state reliability, and low power consumption. The most common flash types are NOR and NAND flash memories. NOR flash memory is particularly well suited for code storage because it supports execute-in-place (XIP) functionality, while NAND flash memory is commonly used for data storage because of its high density and low price [1]. According to an IDC report in 2005 [2], the price of 512Mbit NAND flash memory is now only 15% of NOR flash memory.

As the applications in mobile consumer devices grow in their code and data size with multimedia functionality, NAND flash memory is rapidly replacing NOR flash memory even for code storage. To use NAND flash memory as code storage, the *shadowing* technique is required. Because NAND flash memory does not support XIP, its contents must be copied to RAM during the boot process, and the process is called *shadowing*. However, it causes two side effects: increased RAM usage and delayed boot time. Especially, RAM usage is critical because it has a deep relationship with production costs and power consumption. Moreover, [3] shows that

A. Levi et al. (Eds.): ISCIS 2006, LNCS 4263, pp. 677 – 686, 2006.

RAM price increases exponentially by its capacity. The predicted price of 512Mbits DRAM is $3.62 while the price of 1Gbits DRAM is $22.64 in 2006. Therefore, RAM saving solutions can be critically required for extremely small mobile devices.

Mobile phone systems are facing up to the problem of insufficient memory. Recent featured phones have 64Mbytes RAM, but it is not sufficient because they are increasingly absorbing various functions from other devices such as MP3 players, digital camcorders, and car navigation systems.

We can consider the virtual memory technique of conventional operating systems [4] to reduce RAM usage instead of *shadowing*, and it can expand memory capacity with secondary storage, virtually. However, we cannot use the virtual memory based operating system such as Embedded Linux for mobile phones because the communication protocol stack codes of mobile phone can not be ported to non-RTOS. Therefore, it is necessary to figure out our own demand paging component solution which can be applied to existing RTOS based applications without interference of real-time task execution.

To apply demand paging on mobile devices using real-time operating system (RTOS) and NAND flash memory, unpredicted execution delays of virtual memory systems must be handled in order not to break time dead-lines of real-time tasks. Then, in order to get better performance, the physical characteristics of NAND flash memory must be considered. NAND Flash memory is relatively fast for reading but writing is slow and complicated because of its erasure operation and wear-leveling requirement.

Obviously, it is impossible to use demand paging without execution delays. Therefore, we tried to figure out a graceful architecture where we can selectively use demand paging. That means, the portion of the image related with real time tasks will be excluded from demand paging in our solution. However, the portion is small compared to non-real time tasks in mobile phone systems, and the situation may be similar in other types of mobile consumer devices, fortunately.

We designed a Virtual-ROM for a component based demand paging solution. As its name implies, it influences only the restricted memory space of a processor, and is read-only. For a developer, it is almost the same as a ROM component except that the access to it may be unpredictably delayed by demand paging process.

Fig. 1 shows the difference between the virtual memory of a conventional operating system and a Virtual-ROM. In a virtual memory system (Fig.1. (a)), a user-process can see only a virtual memory space. If a required page is absent in physical memory at a particular moment, the process must wait until the page is loaded by kernel. Fig. 1(b) shows the example of an RTOS-based system's memory space. Normally, an application can see all of the physical memory space. Sometimes, the address mapping can be remapped with a memory management unit statically, but it does not influence the execution of real-time tasks. Fig.1(c) shows an example of the memory space of the system using Virtual-ROM. The Virtual-ROM influences only the restricted memory region. A user can see both physical memory space (outside the Virtual-ROM) and virtual memory space (inside the Virtual-ROM). We can guarantee that Virtual-ROM does not influence real time task execution by excluding the code and data related with real-time tasks from the Virtual-ROM.

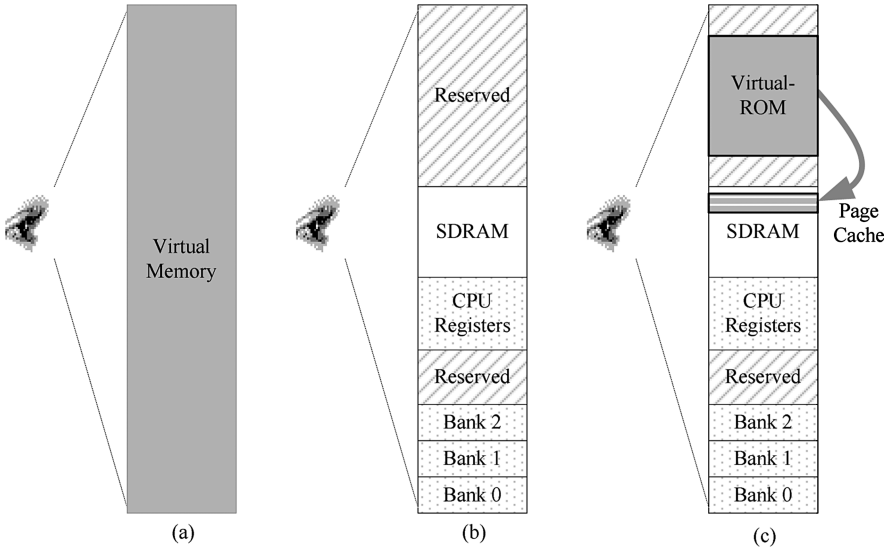

Fig. 1. Examples of the memory spaces of (a) Virtual Memory system, (b) RTOS system, and (c) Virtual-ROM applied RTOS system

Another advantage of Virtual-ROM is its portability. Because Virtual-ROM controls restrict page translation table entries, which are related to its address region, it can be applied to any RTOS that does not change virtual to physical address mapping dynamically for a Virtual-ROM region.

In order to realize our innovative idea on Virtual-ROM, we implemented a Virtual-ROM component for an ARM9 cored processor, and applied it to a commercial CDMA mobile phone. With Virtual-ROM, we were able to successfully save about 30% RAM space. For performance evaluation, the Virtual-ROM was compared with NOR flash memory by trace driven simulation. Extensive simulations with twenty-six traces from SPEC CPU2000 benchmarks [5], show that the overall performance of Virtual-ROM is similar to the performance of NOR flash memory with 70ns access speed.

The rest of the paper is organized as follows: Section 2 outlines the related works in the area of virtual memory and NAND flash memory, and section 3 describes Virtual-ROM design and structure. Section 4 describes the evaluation method and the results briefly, and Section 5 concludes.

2 Related Work

Because of the importance and long history of virtual memory, many researchers have studied it, and many noticeable results have been presented. The virtual memory

concept was introduced in the 1950s. The designers of the Atlas Computer at the University of Manchester invented it to eliminate looming programming problems: planning and scheduling data transfers between main and secondary memory and recompiling programs for each change in the size of the main memory [6]. After Atlas, virtual memory has been used in most operating systems including MULTICS, UNIX, Linux, and Windows systems.

In the 1970s and 1980s, many studies focused on virtual memory systems. Leon proposed a look-ahead paging technique to get better performance [6], and Denning proposed a working set model for program behavior to get optimized performance in a multi-process environment [7]. Rajaraman defined the primitives of virtual memory that affect system performance, and reported a simulation based experimental result of a model of a virtual memory computer [8]. Alok studied the behavior of algorithms for virtual memory and modeled various virtual memory algorithms mathematically [9].

However, most previous research was done on enterprise systems based on a hard disk rather than embedded systems based on NAND flash memory. Because NAND flash memory has very different characteristics from a hard disk, the method and effect of the virtual memory change in a NAND flash memory based virtual memory system.

There have been some researches on NAND flash memory based virtual memory techniques for mobile devices. Park et al. proposed a NAND XIP architecture [10] to emulate the functionalities of NOR flash memory with NAND flash memory, SRAM, and additional hardware logic. They tried to replace NOR flash memory physically with their hardware based solution by using demand paging technique for the purpose. In the NAND XIP architecture, the demanded pages are read from NAND flash memory to the internal SRAM by hardware logic. During the demand paging process, the CPU is pended by a CPU wait signal. NAND XIP is very similar to Virtual-ROM because both are for ROM emulation with NAND flash memory to replace NOR flash memory. However, the methods of approaching the problem are different. While Virtual-ROM emulates ROM with a memory management unit and an exception mechanism, NAND XIP uses additional hardware logic and additional SRAM. NAND XIP is more powerful because of its hardware, but Virtual-ROM is cheaper and more flexible because it needs no additional hardware logic.

NAND flash memory based demand paging in Embedded Linux has also been researched [11]. An energy-aware demand paging technique to lower the energy consumption of embedded systems was presented. It considered the characteristics of the interactive embedded applications with large memory footprints. In the research, they showed that a NAND flash memory based demand paging system consumed less energy than systems based on a shadowing technique. They also proposed a flash memory aware page replacement policy, named Clean-First LRU, which can reduce the number of write and erase operations in NAND flash memory. The algorithm gives a higher priority to clean pages than dirty pages in selecting a victim page because the cost of writing is much higher than the cost of reading in NAND flash memory.

For low-end embedded systems without memory management units, Park et al. proposed a compiler-assisted demand paging technique [12]. In this research, they proposed a kind of automatic overlay technique with a compiler post-pass analysis of an ELF executable image and page manager. This research is similar to Virtual-ROM because both methods are for read-only code and data in NAND flash memory. But the baseline was different. Compiler-assisted demand paging is for systems that do not have a memory management unit, but Virtual-ROM is for systems that have a memory management unit. Because Virtual-ROM uses hardware memory manage unit, it is much faster and easier to use than compiler-assisted demand paging. However, Virtual-ROM cannot be applied to systems that do not have a memory management unit.

There have also been trials to change the traditional virtual memory framework. Kieran and David proposed a new method to take over page replacement in the operating system virtual memory implementation [13]. In the method, page-cache management was done by application. However, unlike our approach most parts still remain as a roll of the operating system. Our Virtual-ROM is completely independent of a particular operating system.

3 Virtual-ROM

Virtual-ROM is designed as a standalone component to maximize portability. Fig. 2 shows the internal structure of the Virtual-ROM component. It consists of three modules: the *Interface module*, the *Fault Handling module*, and the *Page Cache Management module*. The *Interface module* initializes the page translation table, and installs the exception handler for the Virtual-ROM component. The *Fault Handling module* handles page faults at run time, and the *Page Cache Management module* manages the page cache with its page replacement algorithm.

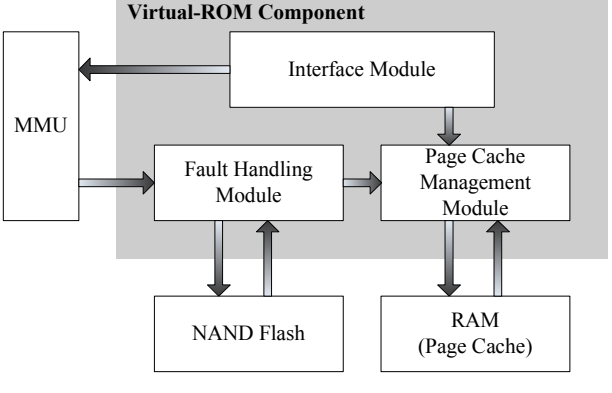

Fig. 2. Virtual-ROM component internal structure: Virtual-ROM consists of three internal modules: the Interface module, Fault Handling module, and Page Cache Management module

3.1 Virtual-ROM Activation Process

Virtual-ROM must be activated after the memory management unit is initialized. Even in RTOS, the memory management unit is commonly used to control cache on/off, to remap memory mapping statically, and sometimes to protect memory. Because the Virtual-ROM also has to use the memory management unit, the Virtual-ROM must be activated after the RTOS board support package (BSP) initializes the memory management unit. Virtual-ROM requires the partial memory space of a processor, and does not influence the outside of the Virtual-ROM region. Because the level 1 page translation table is mandatory to enable the memory management unit, the RTOS BSP builds the page translation table up to the platform's memory configuration during the boot process.

The Virtual-ROM activation API builds a level-2 page translation table to use small mapping units such as 4Kbytes and 1Kbytes pages. To use the two-level page translation, it finds the existing level-1 page translation table's base address and modifies its entries, which are associated with the Virtual-ROM region. **Fig. 3**(a) shows the status when a Virtual-ROM component completes the building of the page translation table.

After building the page translation table for the Virtual-ROM region, it installs its own prefetch and data abort exception handlers. The page fault exception is the core mechanism which is supported by the hardware memory management unit for demand paging. The Virtual-ROM component hooks the existing exception handlers, as shown in **Fig. 3**(b), to catch the CPU control for page fault exceptions. The Virtual-ROM's exception handler checks whether a page fault occurs in the Virtual-ROM region. If the exception occurs outside the Virtual-ROM region, it bypasses the exception to the original handler. The *Fault Handling module* in **Fig. 2** includes the Virtual-ROM's exception handlers. Finally, the Virtual-ROM *Interface module* allocates heap memory for the page cache, and passes it to the *Page Cache Management module* to initialize its data structure by calling the initialization API of the module.

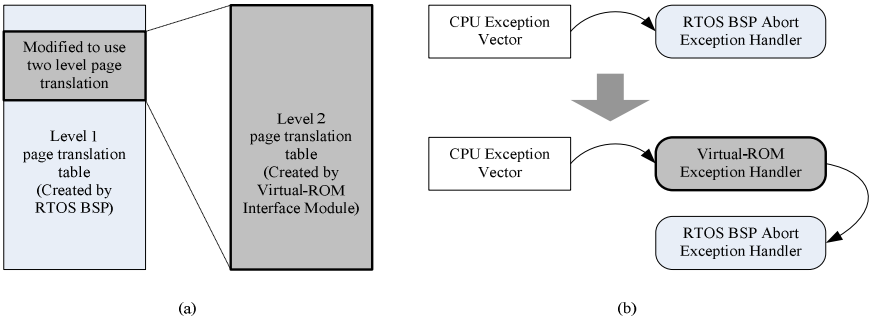

Fig. 3. Virtual-ROM activation process: (a) Modification of the level 1 page translation table, and creation of the level 2 page translation table. (b) The exception handlers of the Virtual-ROM are installed to catch the page translation aborts.

The following code example shows the activation process of the Virtual-ROM in Nucleus OS environment. As shown in the below code example, the Virtual-ROM must be activated after the memory management unit is initialized.

```
/* Nucleus C level initialization. It is called from
   int_pid.s after stack initialization */

void * BoardInit(void *pFreeMem)
{
    __rt_lib_init();
    GPIOInit();
    InitDbgSerial(115200);
    InitHWTimer();

    pFreeMem = MMU_Init(pFreeMem);

    VROM_Init(0x60000000, /* Virtual-ROM Base Address */
              0x01700000, /* Virtual-ROM Size (23MB)  */
              pFreeMem,   /* Page Cache Base Address  */
              0x00500000, /* Page Cache Size   (5MB)  */
              PART_ID_VROM) /* NAND partition ID      */

    pFreeMem += 0x00500000;
    pFreeMem = LCDInit(pFreeMem);
    TimerInit(TICK2SEC);

    return pFreeMem;
}
```

3.2 Page Fault Handling Process in Virtual-ROM

When a page fault occurs, the exception handler of *Fault Handling module* is called. It finds out the exception address, and checks whether the fault occurs inside the Virtual-ROM region. If the fault occurs outside the Virtual-ROM region, it bypasses the fault to the original exception handler of the system. Otherwise, it services the page fault. Then, it requests the *Page Cache Management module* to make a free page frame. Because the Virtual-ROM is read-only, the contents on the victim page can just be flushed without write-back. Then, it reads the required page from the NAND flash memory, and changes the page translation table entry. **Fig. 4** shows the procedure.

Because the ARM processor does not have any hardware support of reference bit for the LRU page replacement algorithm, the Virtual-ROM uses an LRU approximation algorithm: a Clock algorithm with two arms [14]. A pseudo-fault technique is used to sense the page access pattern. The pseudo-fault is not a real fault, so page reading form NAND flash memory is not necessary.

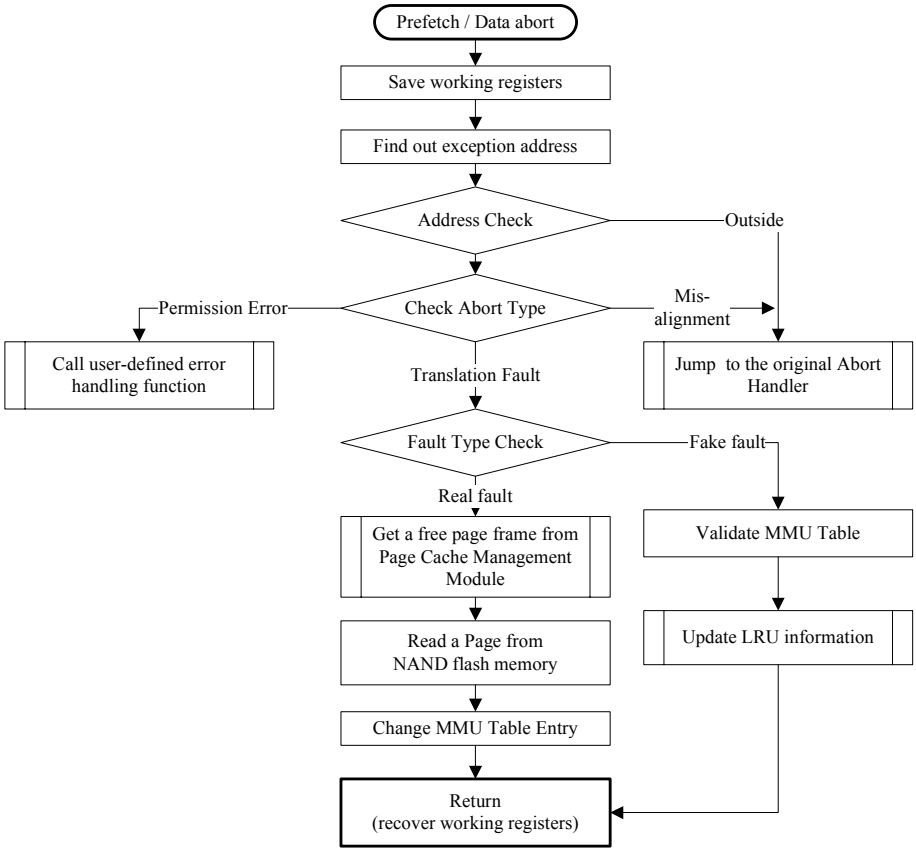

Fig. 4. The page fault handling process in the Virtual-ROM component: The system checks whether the exception occurs inside the Virtual-ROM, and then services the exceptions

4 Evaluation

To evaluate the Virtual-ROM performance, we performed a trace-driven simulation with SPEC 2000 CPU benchmark code access traces. We measured the required parameters from the real implementation of Virtual-ROM on ARM920T core evaluation board for the simulation. The page fault handling time was about 130us, 190us, and 310us for 1Kbyte, 2Kbyte, and 4Kbyte page configurations, respectively.

The traces were obtained from the public Trace Distribution Center [15]. A total of 100 millions traces for each benchmark of the SPEC CPU 2000 (CINT 2000 12 benchmarks, 14 CFP2000 benchmarks) was available. Because the Virtual-ROM is read-only, we extracted only the code traces, and about 65% of the traces were extracted. We simulated the code accesses with the traces, and calculated the time took for all traces. **Fig. 5** shows our simulation results. The results show that the 1K paged Virtual-ROM is comparable with 70ns NOR flash memory.

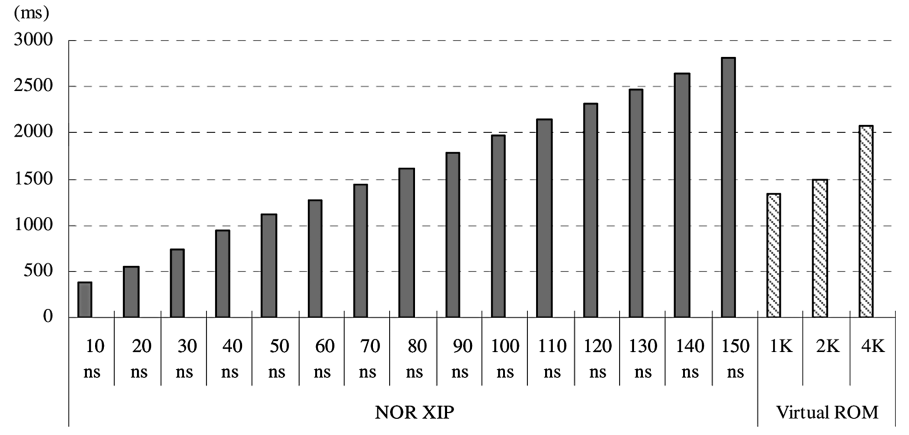

Fig. 5. Trace driven simulation result. The simulator calculates access time for the given address traces. The result above shows the total sum of all simulation results with 26 SPEC CPU 2000 benchmarks.

We applied the Virtual-ROM to a CDMA mobile phone with an ARM926EJ processor, 384Mbytes NAND flash memory, and 64Mbytes SDRAM. Previously, the phone used 29Mbytes RAM for shadowing purposes. We divided 29Mbytes of shadowing image into two partitions: a 6Mbyte shadowing partition that includes real-time tasks and 23Mbytes of Virtual-ROM partition. Because we gave 5Mbytes RAM for the page cache for the Virtual-ROM, we were able to save 18Mbytes of RAM successfully.

5 Conclusion

In this paper, we have proposed an RTOS based demand paging component, a Virtual-ROM to save shadowing RAM in NAND flash memory based mobile devices. Virtual-ROM enables a developer to use virtual and physical memory at the same time. Moreover, its simple but efficient design with a minimized porting cost makes it easier to apply to existing mobile applications.

Our trace driven simulation shows that the performance of the Virtual-ROM is comparable with 70ns NOR flash memory for the traces from SPEC CPU 2000 26 benchmarks. Furthermore, we were able to save about 30% of RAM from CDMA mobile phones with Virtual-ROM resulting in a significant reduction in production costs.

Further research is required to determine which code should be excluded from the Virtual-ROM region for real-time guarantee. For this purpose, we are planning to develop a methodology that includes a run-time profiler and a static image analyzer.

References

1. M-Systems, Two Technologies Compared: NOR vs. NAND. White Paper, 91-SR-012-04-8L, Rev 1.1, (Jul. 2003).
2. Takuya Inoue, Mario Morales, and Soo-Kyoum Kim, Worldwide Flash Memory 2005-2008 Forecast and Analysis. In February 2005, IDC #32854, Vol. 1.
3. Soo-Kyoum Kim, and Shane Rau, Worldwide DRAM 2005 - 2010 Forecast, In December 2005, IDC #34658, Vol.1.
4. Peter J. Denning, Virtual Memory. In ACM Computing Surveys (CSUR), Vol. 2 Issue 3 (Sep. 1970), 153 – 189.
5. Standard Performance Evaluation Corporation, SPEC CPU 2000 V1.3, http://www.spec.org/osg/cpu2000/
6. Leon R. Wechsler, The effect of look-ahead paging in a virtual memory system as determined by simulation. In Proceedings of the 1st symposium on Simulation of computer systems, (1973), 234 – 241.
7. Peter J, Denning, The working set model for program behavior. In Communications of the ACM, Vol.11, Issue 5 (May 1968), 323 – 333.
8. M. K. Rajaraman, Performance of a virtual memory: some experimental results. In ACM SIGMETRICS Performance Evaluation Review, Vol. 8, Issue 4, (1979), 63 – 68.
9. Alok Aggarwal, Virtual memory algorithms. In Proceedings of the twentieth annual ACM symposium on Theory of computing, (1988), 173 – 185.
10. Chanik Park, Jaeyu Seo, Sunhwan Bae, Hyojun Kim, Shinhan Kim, and Bumsoo Kim, A low-cost memory architecture with NAND XIP for mobile embedded systems. In Proceedings of CODES+ISSS'03 (Oct. 2003), 138 – 143.
11. Chanik Park, Jeong-Uk Kang, Seon-Yeong Park, and Jin-Soo Kim, Energy-aware demand paging on NAND flash-based embedded storages. In Proceedings of the 2004 international symposium on Low power electronics and design (Aug. 2004), 338 – 343.
12. Chanik Park, Junghee Lim, Kiwon Kwon, Jaejin Lee, and Sang Lyul Min, Compiler-assisted demand paging for embedded systems with flash memory, In Proceedings of the 4th ACM international conference on Embedded software EMSOFT '04 (Sep. 2004), 114 – 124.
13. Kieran Harty, and David R. Cheirton, Application-controlled physical memory using external page-cache management. In Proceedings of the fifth international conference on Architectural support for programming languages and operating systems (1992), 187 – 197.
14. Babaoglu, O. and Joy, W.N., Converting a swap-based system to do paging in an architecture lacking page-referenced bits. In Proceedings of Eighth ACM Symposium on Operating Systmes Principles (Dec. 1981), 78 – 86.
15. Performance Evaluation Laboratory of Brigham Young University, Trace Distribution Center, http://tds.cs.byu.edu/tds/index.jsp