Energy-Aware Demand Paging on NAND Flash-based Embedded Storages*

Chanik Park ci.park@samsung.com
Jeong-Uk Kang {ux,parksy}@camars.kaist.ac.kr
Seon-Yeong Park
Jin-Soo Kim jinsoo@cs.kaist.ac.kr

Memory Division
Samsung Electronics Co.

Computer Science Division
Korea Advanced Institute of Science and Technology (KAIST)

ABSTRACT

The ever-increasing requirement for high-performance and huge-capacity memories of emerging embedded applications has led to the widespread adoption of SDRAM and NAND flash memory as main and secondary memories, respectively. In particular, the use of energy consuming memory, SDRAM, has become burdensome in battery-powered embedded systems. Intuitively, though demand paging can be used to mitigate the increasing requirement of main memory size, its applicability should be deliberately elaborated since NAND flash memory has asymmetric operation characteristics in terms of performance and energy consumption.

In this paper, we present energy-aware demand paging technique to lower the energy consumption of embedded systems considering the characteristics of interactive embedded applications with large memory footprints. We also propose a flash memory-aware page replacement policy that can reduce the number of write and erase operations in NAND flash memory. With real-life workloads, we show the system-wide Energy•Delay can be reduced by 15∼30% compared to the traditional shadowing architecture.

Categories and Subject Descriptors

B.3.2 [**Memory Structures**]: Design Styles—*Virtual memory*; D.4.2 [**Operating Systems**]: Storage Management—*Secondary storage, Virtual memory*; C.4 [**Performance of Systems**]: Design Studies

General Terms

Algorithms, Measurement, Performance, Design

Keywords

Demand paging, Page replacement, Virtual memory, NAND flash memory, Embedded systems, Embedded storages

*This work was supported in part by University IT Research Center Project.

ISLPED'04, August 9–11, 2004, Newport Beach, California, USA.

1. INTRODUCTION

As emerging embedded applications grow in code and data requirement, there has been more need for increased memory size, usually SDRAM. Recently, for huge capacity, flash memory is used as a secondary storage device due to its versatile features: non-volatility, solid-state reliability, low power consumption, etc. The most popular flash types are NOR and NAND. NOR flash is particularly well suited for code storage and execute-in-place (XIP) applications, which require high-speed random access. While NAND flash memory provides high density and low-cost data storage, it does not lend itself to XIP applications due to the sequential access architecture and long random access latency.

Current design practice for satisfying the strong requirement of high-performance and large capacity is to transfer OS and application code from the non-volatile memory device such as NAND flash memory to the volatile memory such as SDRAM, which is hereafter referred to as memory "shadowing" (Figure 1(a)). The shadowing offers the best performance possible since all the code is executed in fast SDRAM memory, but copying overhead at boot-time contributes to the slow boot process. In addition, a large amount of SDRAM is necessary to hold both OS and application codes even though all applications are not executed. Most of all, high power consumption from power hungry SDRAM memory is a critical problem for battery-operated embedded systems.

Intuitively, in order to utilize the limited memory space, we can take advantage of demand paging technique with the assistance of operating system and memory management unit (MMU) hardware (Figure 1(b)). Demand paging allows application code to be executed by loading code and data on demand from the secondary storage to the main memory.

However, frequent data transfers between the secondary storage and the main memory may require additional energy budget. Furthermore, possible degradation of system performance is another concern to embedded system designers. This problem can be worsened by the operational characteristics of NAND flash memory which will be explained in the next section.

In this paper, we investigate the impact of demand paging on energy consumption and performance. In particular, we focus on NAND flash-based demand paging due to its popularity in mobile embedded systems. This is the first approach using demand paging on NAND flash-based storage architecture.

Our contributions are as follows:

- We characterize the interactive applications with large

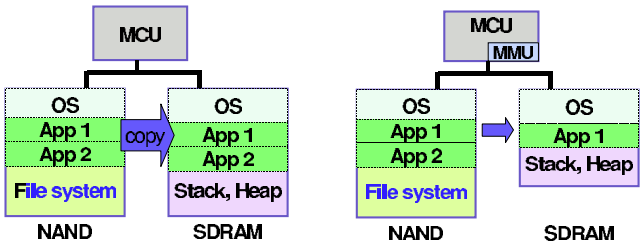

Figure 1: Memory architectures for mobile embedded systems

memory footprints in terms of memory access pattern and CPU utilization.

- We analyze the performance and energy consumption bottleneck in demand paging.
- We present an improved page replacement algorithm (called CFLRU) which is optimized for NAND flash memory.
- We evaluate the Energy•Delay metric of each application on two memory architectures with system-level energy modeling.
- Finally, we present cost and energy efficient embedded storage architecture based on NAND flash memory.

The rest of this paper is organized as follows. The next section describes the characteristics of NAND flash memory. In Section 3, we present the effectiveness of demand paging on NAND flash memory and our modified page replacement algorithm. In Section 4, we show experimental results with trace-driven simulation environment and evaluate performance and power consumption over real-life workloads. Section 5 reviews related work. Finally, our conclusions and future work are drawn in Section 6.

2. BACKGROUND

A NAND flash [12] memory consists of a fixed number of blocks, where each block has 32 pages and each page consists of 512 or 2048 bytes main data and 16 or 64bytes spare data. Read/write operation is performed on page basis. In order to read a page, command and address are given to NAND flash memory through I/O pins. After a fixed delay time of 10∼25μs, the selected page is loaded into data and spare registers in NAND flash. Hereafter 8bit or 16bit data can be sequentially fetched from data and spare registers every 50ns. Table 1 shows the characteristics of various non-volatile storage devices. In case of NAND flash memory, differently from MicroDrive [1], the write operation requires relatively long latency compared with the read operation. As write operation usually accompanies erase operation, the operational latency becomes longer. This also results in different energy consumption aspect depending on operations.

Currently the operating system implementation of demand paging is customized for hard disk drive to hide the long seek and rotational times. On the contrary, NAND flash has advantage in fixed operation times. However, it has disadvantage in updating a page since a page to be updated should be erased before write operation is performed. Another limitation is that the number of updating a block is limited to about 100,000 times. Thus, the number of erase and write

Table 1: Characteristics of non-volatile devices

Device	Current (mA)		Access Time (4KB)		
	Idle	Active	Read	Write	Erase
NOR	0.07	40	23μs	28ms	1.2sec
NAND	0.05	20	156μs	652μs	2ms
MicroDrive	20	250	21ms	21ms	N/A

(Parameters are obtained from [12, 5, 1])

operations should be minimized to not only reduce the energy consumption but also lengthen the endurance of NAND flash memory. In this paper, we propose a modified page replacement policy to minimize the number of erase and write operations. The detailed technique will be explained in later sections.

3. DEMAND PAGING FOR NAND FLASH MEMORY

Demand paging is a virtual memory technique used by modern operating systems, where code or data are loaded from the secondary storage only as needed by a process. Previously, demand paging has not been used extensively for embedded systems mainly because memory management unit (MMU), a hardware required to support demand paging, adds to costs in embedded processors. However, many recent embedded processors, such as Motorola 68060, Intel PXA255, ARM 920T, and PowerPC 440GX, come fully equipped with MMU in order to ease memory management.

We predict that demand paging (more generally the virtual memory system) will prevail also for embedded systems in the near future due to the following reasons. First, as the performance of embedded processors improves, there will be a growing demand for performing complex tasks, which inevitably increases the application's memory footprints and the required main memory size. Secondly, it is increasingly common to download new applications from the network for PDAs or hand-held phones, in which case the number of applications and their total memory requirement are changing dynamically. Finally, it is not efficient, in terms of energy consumption and space utilization, to keep all the applications in the main memory as is done in shadowing, because they need not be active at the same time. The aforementioned situations can be handled elegantly using demand paging, as we can keep only the working sets of active processes in the main memory, while backing up the rest in the secondary storage.

3.1 Impact on Energy Consumption

We consider, in this paper, using NAND flash memory as the backing store for demand paging. By moving unused or least recently used pages into NAND flash memory, it is apparent that we can reduce the main memory size and hence the SDRAM retention energy ($E_{SDRAM\_Retention}$) caused by the main memory. This is because the retention energy of NAND flash memory is negligible compared to that of SDRAM. However, such energy saving does not come without a price; it is necessary to spend a certain amount of energy (E_{NAND}) for accessing NAND flash memory during swap in/out.

One can easily see that $E_{SDRAM\_Retention}$ depends on the total execution time of the application, while E_{NAND} only depends on the amount of memory references when CPU is active. Therefore, the actual benefit of using demand paging

Table 2: Workloads used in the experiments and their characteristics

Workload	CPU utilization	Memory used (MB)	Memory references		Scenario
			total	read : write	
GQVIEW	0.57%	20.77	28,940,881	43% : 57%	Performs a slide show of seven images and adjusts their sizes.
KWORD	1.64%	27.52	4,779,386	81% : 19%	Edits several lines in a text document.
KSPREAD	2.62%	38.93	11,366,261	68% : 32%	Calculates sum, avg., min., and max. of numerical data, and sorts them.
ACROBAT	4.31%	35.79	10,815,848	40% : 60%	Views a PDF document.
MOZILLA	0.77%	38.04	41,533,372	85% : 15%	Browses several web sites including Yahoo, CNN, Hotmail, Amazon, etc.

on NAND flash memory is affected by CPU utilization, i.e., the ratio of the active CPU cycles to the total CPU cycles. The lower CPU utilization suggests that relatively the larger amount of energy is spent for $E_{SDRAM\_Retention}$, thereby increasing the benefit of demand paging.

In order to investigate the typical range of CPU utilizations, we actually ran five interactive applications on Linux and measured their CPU utilizations using the `time` command. Table 2 describes the real-life workloads and scenarios used during the measurement: GQVIEW (an image viewer), KWORD (a word processor), KSPREAD (a spreadsheet application), ACROBAT (a PDF reader), and MOZILLA (a web browser). We can see that all the tested workloads show less than 5% of CPU utilization, with the average of 2%. Zhong and Jha have also reported the similar result that over 95% of the time is spent in waiting for user input on a Linux-based Sharp Zaurus SL-5500 PDA [14]. These results suggest that, for interactive embedded applications, CPU is in idle state for the significant percentage of the total execution time. In this circumstance, demand paging is an effective way to reduce $E_{SDRAM\_Retention}$. We present the actual experimental results in section 4.

Note that demand paging slightly increases the total execution time of the application as a result of servicing page faults. However, under the multi-processing environment, such delay can be hidden by switching to another process.

3.2 Flash Memory-Aware Page Replacement

Traditional operating systems have been optimized for decades under the assumption that the secondary storage consists of magnetic disks. With the advent of NAND flash memory as a viable candidate for the secondary storage, however, we need to revisit the various operating system policies and mechanisms which were developed for disks. In this section, we propose a page replacement policy which is aware of the fact that NAND flash memory is used for swap device instead of disks.

Existing operating systems mainly use LRU (Least Recently Used) or pseudo-LRU page replacement policies, whose primary goal is to minimize the page fault ratio. However, as the write cost for evicting a page is much higher than the read cost for swapping in a page in NAND flash memory, it is sometimes useful to keep dirty pages as long as possible so that the number of swap outs can be minimized. Reducing the number of swap outs decreases not only the average miss penalty for handling page faults, but also the number of erase operations in NAND flash memory.

Suppose pages are recently accessed in the order of A, B, C, D, and E, as illustrated in Figure 2. Under the LRU page replacement algorithm, the sequence of victim pages

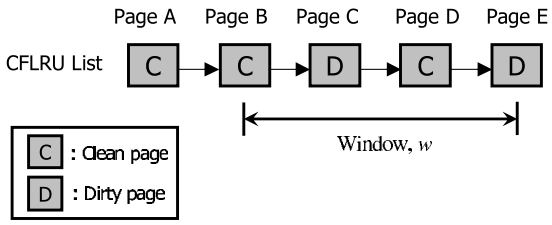

Figure 2: CFLRU page replacement example

is E, D, C, and B, always evicting the least recently used page first. However, when we use NAND flash memory as a swap device, it may be advantageous to evict the clean page D first in order to reduce the number of flash write operations, even though the page is more recently accessed than the dirty page E. We call this CFLRU (Clean First LRU) and propose it as a new page replacement policy for swapping over NAND flash memory.

CFLRU is a modified LRU algorithm, where a clean page is more preferred than a dirty page as a victim. The idea is to search for a victim page in the following order:

1. least recently used clean pages
2. least recently used dirty pages
3. recently used pages (clean or dirty)

As the page fault ratio may increase if we evict the recently used clean page, only the clean pages within the predetermined window size (w) become candidates for a victim in CFLRU. If CFLRU does not find any clean page within the window, it is converted into the normal LRU algorithm. In Figure 2, pages are evicted in the sequence of D, B, E, and C, when we use CFLRU. The performance of CFLRU is discussed in detail in section 4.

4. EVALUATION

4.1 Energy Consumption Modeling

The total energy consumed by the system is the sum of energies consumed by system components such as processor, bus, memory and the DC-DC converter. In this paper, we concentrate on the energy consumption of the processor, memory and system bus.

$$E_{task}(t) = E_{CPU}(t) + E_{mem}(t) + E_{bus}(t) \quad (1)$$

The energy consumption of CPU is divided into active and inactive state depending on whether CPU is in execution or in idle mode:

$$E_{CPU}(t) = t_{active} \times P_{CPU\_active} + t_{inactive} \times P_{CPU\_idle} \quad (2)$$

Table 3: Power and energy coefficient values

Component	Coefficient	Value
CPU [6]	P_{CPU_active}	411 (mW)
	P_{CPU_idle}	0.15 (mW)
SDRAM [11]	E_{read}	17.95 (mJ/4KB)
	E_{write}	13.21 (mJ/4KB)
	P_{idle}	45.1 (mW)
	$P_{refresh}$	6.4 (mW)
	$P_{retention}$	1.8 (mW)
NAND [12]	E_{read}	9.44 (uJ/4KB)
	E_{write}	76.08 (uJ/4KB)
	E_{erase}	16.49 (uJ/4KB)
	$P_{retention}$	0.13 (mW)

As for memory, the number of accesses to the memory (N_{access}) is directly proportional to the energy consumption. The unit access energy consumption (E_{mem_active}) can be estimated from the active power specified in the data sheet [11, 12]. Specifically, SDRAM consumes idle power (P_{idle}) and refresh power ($P_{refresh}$) during the CPU execution. If there is no memory access, the memory stays in the power down state consuming only retention power ($P_{retention}$).

$$E_{mem}(t) = E_{mem\_active} \times N_{access} + t_{active} \times (P_{idle} + P_{refresh}) + t_{inactive} \times P_{retention} \quad (3)$$

In case of bus model, the energy is a function of the voltage swing on the lines that switched (V_{dd}), the capacitance of one interconnect line and the pins attached to it (C_{switch}[1]), the number of bus accesses (N_{access}), the bus width (N_{width}) and the bit change rate R_{change}[2].

$$E_{bus}(t) = R_{change} \times N_{width} \times N_{access} \times C_{switch} \times V_{dd}^2 \quad (4)$$

4.2 Evaluation Methodology

Table 3 shows various coefficient values we use to determine the total energy consumption according to the model presented in section 4.1. We assume CPU is Intel PXA260 400MHz [6] and parameters for SDRAM and NAND are obtained from [11] and [12], respectively. Other information, such as t_{active}, N_{access}, the number of swap ins/outs, etc. is collected from a trace-driven simulation.

The memory reference trace of each application is gathered using Valgrind [9] on Linux/x86 platform under the same scenario shown in Table 2. Valgrind is a tool that uses dynamic binary translation technique and its cache simulation module called Cachegrind has been slightly modified to generate instruction/data reference traces. We have translated the total number of instructions obtained from Valgrind into the corresponding CPU cycles (t_{active}) for PXA260 with the assistance of profiling results using ARMulator. The total execution time t is calculated by dividing t_{active} by CPU utilization.

We compare two memory configurations in the experiments. For shadowing, we assume the system consists of two 32MB SDRAM modules and one NAND flash memory module, while only one 32MB SDRAM module is used for demand paging. When we simulate LRU and CFLRU page replacement algorithms, it is also assumed that Linux operating system and X-windows consume about 16MB and this amount of main memory is not available to applications. Therefore, the memory size that applications can freely use is limited to 48MB for shadowing, and to 16MB for demand paging.

Unless otherwise explicitly stated, it is assumed that CPU utilization is 5%, and the window size (w) for CFLRU is 30% of the total number of physical pages.

[1]Pin capacitance values are obtained from the data sheets [6, 11, 12].

[2]The bus energy is consumed when a value of a line (bit) is changed from 0 to 1 or 1 to 0. In this paper, the bit change rate is determined to be the average number of changed bits per bus width (32bits) through program execution profiling. In our case, the bit change rate value is 0.25.

Table 4: Normalized energy and delay

Workload	Normalized Energy (E)	Normalized Delay (D)	$E \bullet D$
GQVIEW	0.69	1.11	0.77
KWORD	0.68	1.10	0.75
KSPREAD	0.69	1.12	0.77
ACROBAT	0.65	1.07	0.70
MOZILLA	0.74	1.15	0.85

4.3 Experimental Results

In this section, we show our experimental results as a result of exploring two memory architectures: shadowing vs. demand paging. Figure 3 compares the total energy consumption of shadowing and demand paging, giving a breakdown by the energy consumption sources: Bus, NAND flash memory, SDRAM (for Retention and Reads/Writes), and CPU. On the average, we can see that CPU is responsible for about 15% of the total energy consumption, while SDRAM modules for the rest in case of shadowing. The portion of energy consumed by Bus is less than 1% in both cases.

It is not surprising to see that the energy consumption by CPU and SDRAM access does not change much even if we use demand paging. Ideally, demand paging should reduce the energy required for SDRAM retention by half, since there is only half of the SDRAM module. This is not the case in Figure 3, however, because the total execution time is slightly increased due to page faults. Demand paging also consumes some additional energy in NAND flash memory for swap in/out.

Table 4 shows the normalized energy and delay of the demand paging architecture over the shadowing architecture. On the average, about 30% reduction in energy consumption is achieved in case of demand paging architecture, while delay is increased by about 10%. As a result, the Energy•Delay of demand paging is reduced by approximately 23% over that of shadowing. Demand paging overhead is offset by significant energy gain.

Figure 4 describes the relative energy consumption of demand paging over shadowing in proportion to CPU utilization. ACROBAT and MOZILLA are selected due to their best and worst characteristics in terms of swapping overhead, respectively. ACROBAT shows better energy efficiency until CPU utilization reaches 40%, while MOZILLA shows drastic degradation of energy efficiency from the point of 15% in CPU utilization axis. According to our analysis results, most applications show better energy efficiency than in shadowing architecture when CPU utilization is less than 20∼40%.

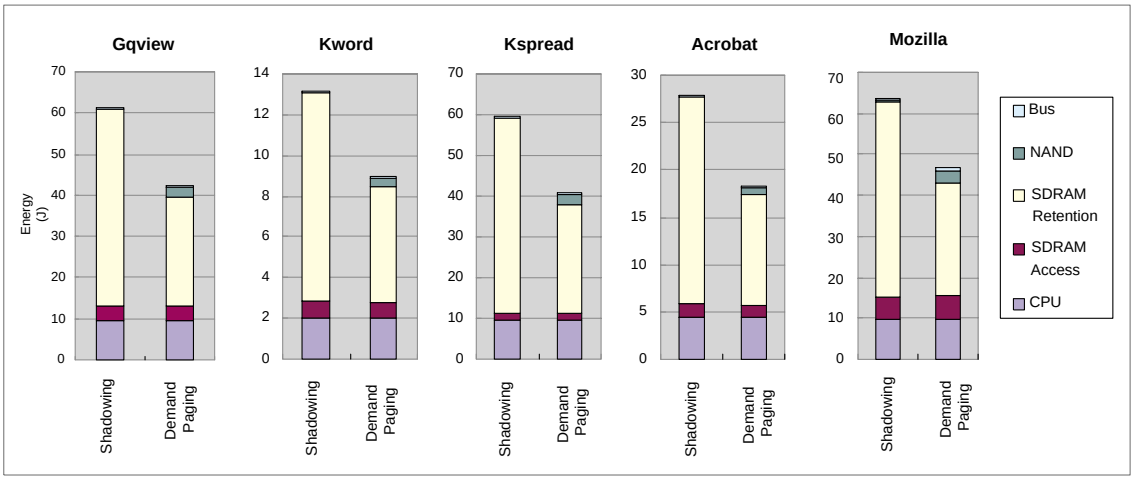

Figure 3: Energy consumption breakdown of each application

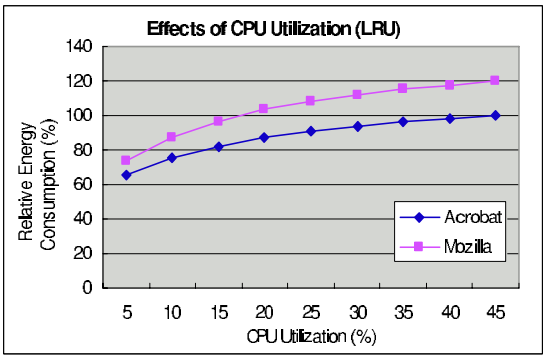

Figure 4: Effects of CPU utilization

In order to compare the performance of LRU and CFLRU, we have analyzed the energy consumption during NAND flash access, namely, swapping in/out operations in Figure 5. As explained in section 3.2, CFLRU is advantageous for reducing the number of swap out operations, while it tends to slightly increase the number of swap in operations since the recently referenced page can be a victim if the page remains in the clean state.

Note that GQVIEW and ACROBAT have little benefit from CFLRU policy. One of the common characteristics in these applications is that they exhibit the relatively large number of memory write operations. From Table 2, we can see that about 60% of memory references are writes in GQVIEW and ACROBAT, while the percentage of memory writes in other applications is less than 30%. As the percentage of memory write operations increases, it will be more and more difficult to find a clean page within the CFLRU's window, which ultimately transforms CFLRU into LRU. When the memory references are mostly reads as in KWORD, KSPREAD, and MOZILLA, CFLRU shows better performance than LRU. For example, the number of swap out operations is reduced from 5369 to 5156 in ACROBAT. The minimization of write and accompanying erase operations also results in improving the endurance of NAND flash memory.

On the other hand, in Figure 5, it is interesting to see that the number of swap in has been noticeably increased in MOZILLA. This is because the working set size of MOZILLA is significantly large compared to other applications. When the application's working set size is large, some of pages that belong to the working set will reside in the CFLRU's window, which forces CFLRU to evict frequently referenced clean pages in order to keep dirty pages. This implies that the window size in CFLRU should be dynamically adjusted according to the application's working set size, but we leave it to future work.

Figure 6 shows the change of the normalized Energy•Delay value in proportion to CPU utilization. We selected the MOZILLA trace since it shows the worst case demand paging performance due to its large working set size. We can see CFLRU shows gradual increase in Energy•Delay compared with the LRU policy. Above 10% of CPU utilization in Figure 6, both LRU and CFLRU show the increase in Energy•Delay. This arises from the fact that we consider a single process execution assuming that CPU is in idle state during swap in/out operations. In the multi-processing environment, however, CPU can be more utilized through context switching to another ready process. The performance impact of demand paging in the multi-processing environment remains as a future work.

5. RELATED WORK

In the literature, a little research work is found in related to our approach. Lee et al. [8] presented energy-aware memory allocation schemes in heterogeneous non-volatile memory systems. They characterized the energy consumption coefficient over various memory components including retention energy that is indispensable in SDRAM memory. Even though its approach is similar to ours in that it considered retention energy and energy-aware allocation, they do not address dynamic loading effect on system energy consumption considering operating system.

Zhong et al. [14] analyzed the power characteristics of interactive systems using four commercial applications shipped with PDA. They say that over 90% of system energy and time was spent waiting for user input. Though our work shares the same observation for interactive applications' behavior, our energy minimization technique is different from theirs in that they utilized user delays for DPM/DVS techniques to reduce system energy consumption very effectively.

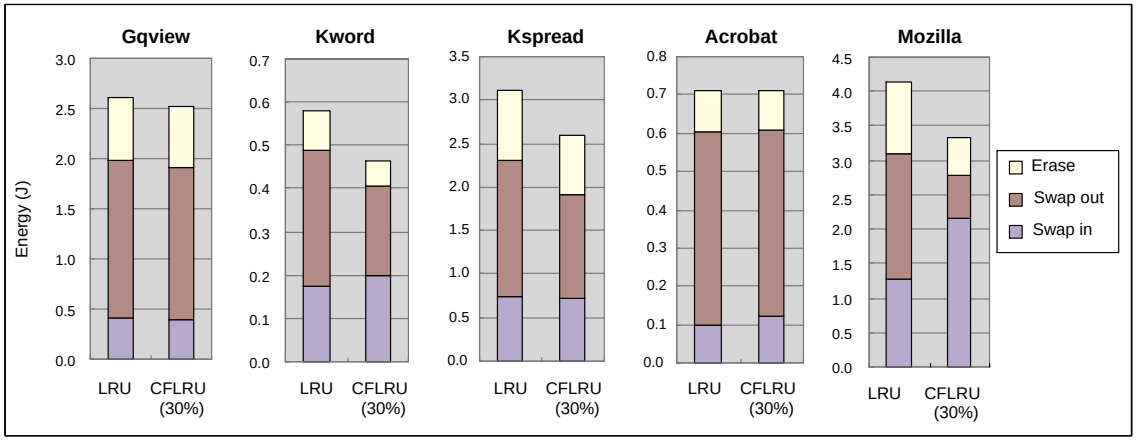

Figure 5: Energy consumption comparison between LRU and CFLRU for NAND access

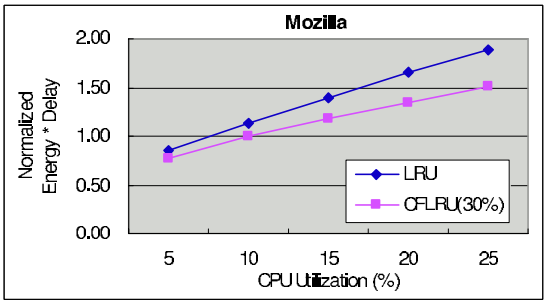

Figure 6: Normalized Energy • Delay for MOZILLA

Recently, Park et al. [10] demonstrated that NAND XIP is a feasible solution even in multimedia applications with soft real-time constraints. They took advantage of cache controller to dynamically load the code page on demand. Even though they show that NAND XIP is applicable to multimedia applications, their approach is limited to code only. However, we cover data and code storage as demand paging target.

As for main memory energy reduction, Lebeck et al. [7] studied OS support for exploiting the power management feature offered by a new power-aware DRAM such as Rambus RDRAM. Delaluz et al. [3] presented a scheduler-based power-management policy to power down unused DRAM banks using the bank usage table of executing processes. In [4] and [2], page allocation algorithm was improved to minimize the number of used banks and data migration in multi-bank memory systems are described, respectively.

In the early Linux 2.4 kernels [13], not recently used (NRU) pages are classified into the inactive dirty list and the inactive clean list, and pages on the inactive clean list can be immediately reused by the page allocation code. Unlike CFLRU, however, the main objective of differentiating inactive dirty pages from inactive clean pages is to delay and accumulate dirty page writes in order to achieve better disk I/O scheduling.

6. CONCLUSIONS

In this paper, we investigated the usefulness of the energy-aware demand paging technique on NAND flash based embedded storages. Through the CPU utilization analysis of interactive applications, we showed the retention energy of SDRAM overpasses the demand paging overhead. Furthermore, we proposed an improved page replacement policy, CFLRU, which takes not only energy consumption saving, but also the life cycle of NAND flash memory into consideration. As a result, we presented the demand paging architecture with NAND flash memory can be an energy and cost efficient solution for ever-increasing memory requirement in mobile embedded systems.

In the near future, we plan to extend CFLRU, our flash memory-aware page replacement policy, and to evaluate the effectiveness of demand paging for multi-processing environment on real platforms.

7. REFERENCES

[1] http://www.flash-memory-store.com/ibmmicrodrives.html.

[2] V. Delaluz, M. Kandemir, and I. Kolcu. Automatic data migration for reducing energy consumption in multi-bank memory systems. In *Proc. DAC*, 2002.

[3] V. Delaluz et al. Scheduler-based DRAM energy management. In *Proc. DAC*, 2002.

[4] H. Huang, P. Pillai, and K. G. Shin. Design and implementation of power-aware virtual memory. In *Proc. USENIX Annual Technical Conference*, 2003.

[5] Intel Corp. Intel flash memory data sheets and specification updates. http://www.intel.com/design/flash/datashts/index.htm.

[6] Intel Corp. *Intel PXA26x Processor Family Electrical, Mechanical, and Thermal Specification Datasheet*. 2004.

[7] A. R. Lebeck et al. Power aware page allocation. In *Proc. ASPLOS*, pages 105–116, 2000.

[8] H. Lee and N. Chang. Energy-aware memory allocation in heterogeneous non-volatile memory systems. In *Proc. ISLPED*, 2003.

[9] N. Nethercote and J. Seward. Valgrind: A program supervision framework. *Electronic Notes in Theoretical Computer Science*, 89(2), 2003.

[10] C. Park et al. A low-cost memory architecture with NAND XIP for mobile embedded systems. In *Proc. CODES+ISSS'03*, pages 138–143, 2003.

[11] Samsung Electronics. Mobile SDRAM (K4S56163LC) data sheets, Dec. 2002.

[12] Samsung Electronics. NAND flash memory & SmartMedia data book, 2002.

[13] R. van Riel. Page replacement in Linux 2.4 memory management. In *Proc. USENIX Annual Technical Conference*, 2001.

[14] L. Zhong and N. K. Jha. Dynamic power optimization of interactive systems. In *Proc. International Conference on VLSI Design*, 2004.