Demand Paging for OneNAND^{TM} Flash eXecute-In-Place

Yongsoo Joo
Seoul National University
Korea
ysjoo@cslab.snu.ac.kr

Yongseok Choi
Seoul National University
Korea
yschoi@cslab.snu.ac.kr

Chanik Park
Samsung Electronics
Korea
ci.park@samsung.com

Sung Woo Chung
Korea University
Korea
swchung@korea.ac.kr

Eui-Young Chung
Yonsei University
Korea
eychung@yonsei.ac.kr

Naehyuck Chang[*]
Seoul National University
Korea
naehyuck@snu.ac.kr

ABSTRACT

NAND flash memory can provide cost-effective secondary storage in mobile embedded systems, but its lack of a random access capability means that code shadowing is generally required, taking up extra RAM space. Demand paging with NAND flash memory has recently been proposed as an alternative which requires less RAM. This scheme is even more attractive for OneNAND flash, which consists of a NAND flash array with SRAM buffers, and supports eXecute-In-Place (XIP), which allows limited random access to data on the SRAM buffers.

We introduce a novel demand paging method for OneNAND flash memory with XIP feature. The proposed on-line demand paging method with XIP adopts finite size sliding window to capture the paging history and thus predict future page demands. We particularly focus on non-critical code accesses which can disturb real-time code.

Experimental results show that our method outperforms conventional LRU-based demand paging by 57% in terms of execution time and by 63% in terms of energy consumption. It even beats the optimal solution obtained from MIN, which is a conventional off-line demand paging technique by 30% and 40% respectively.

Categories and Subject Descriptors

B.3.2 [**Memory Structures**]: Design Styles - Virtual memory; D.4.2 [**Operating Systems**]: Storage Management - Secondary storage

General Terms

Algorithms, Design, Measurement, Performance

[*]Corresponding author

This work is supported by IT R&D Project funded by Korean Ministry of Information and Communications. The ICT at Seoul National University provides research facilities for this study.

Keywords

Demand paging, Embedded systems, NAND flash memory, OneNAND, Page replacement, Virtual memory, XIP

1. INTRODUCTION

Today, a wide range of portable multimedia embedded systems are available to many different categories of end user. Thanks to the continuing evolution of semiconductors, it is not difficult to achieve the functionality and performance necessary for portable multimedia applications. However, cost-effectiveness is still a critical factor in determining profitability for a manufacturer.

Most low-cost embedded systems deploy on-chip SRAM as their main memory and NAND flash as secondary storage. The amount of on-chip SRAM significantly affects the cost of the chip, and thus optimization of the SRAM footprint is an important issue. A typical configuration uses from dozens to a few hundred KB of SRAM, which is mostly dedicated to key functions such as a media decoder. Better interfaces and support for DRM (digital rights management) has recently become mandatory even for low-cost embedded systems. Therefore, the demand for more SRAM grows rapidly, which is a severe challenge for cost-effective design.

Although the use of off-chip SDRAM can resolve the memory requirement, this approach is not suitable for low-cost systems because off-chip memory dramatically increases power consumption, system complexity and raises other issues such as signal integrity, which result in a significant increase in running costs. For these reasons, demand paging with small size on-chip SRAM and NAND flash [1][2][3] used to be a popular way to cope with the limited-size main memory available with 8-bit microprocessors. Now it is being reintroduced to segregate real-time tasks, such as a media decoder, which are pinned on the on-chip SRAM, and non-real-time tasks such as the user interface and DRM, which are loaded from the NAND flash to the SRAM on demand. Fig. 1 shows a typical NAND flash demand paging architecture.

Suppose the footprint of an audio codec is 100KB and its DRM footprint is 200KB. Old MP3 players with a basic user interface and no DRM can function with a 128KB on-chip SRAM, but at least 384KB is required to support these extra features. Alternatively, NAND flash and demand paging can be used to support a better interface and DRM, and the requirement for on-chip SRAM can still be kept at 128KB, although the performance will be lower than a 384KB systems. This is a good compromise because multimedia performance will be the same because the audio codec remains on the on-chip SRAM.

The interface to NAND flash is asynchronous and sequential, and so there are severe delays from the demand paging system whenever there is a page fault in demand paging systems. Recently, Samsung Semiconductor announced a new hybrid memory

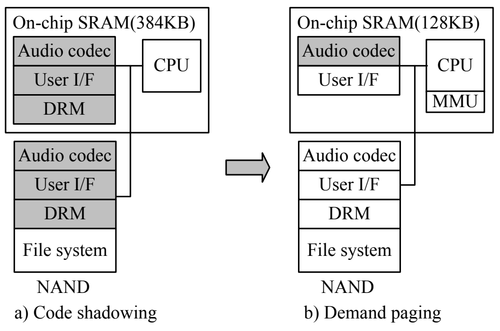


Figure 1: Different memory architectures in media players.

that combines SRAM and NAND flash, called OneNAND [4][5]. In OneNAND the page register is replaced with dual random-accessible SRAM buffers, which support simultaneous page loading from the flash array to the buffer, and data transfer between the buffer and the on-chip SRAM. In addition, the use of the SRAM buffers gives OneNAND another feature, XIP (eXecute-In-Place) capability, which can be utilized for initial boot-up sequence which eliminates the need for NOR flash, although the capability is limited to the SRAM buffers. In general, programs residing in secondary storage devices need to be copied into main memory before execution; but some secondary storage devices such as ROMs and NOR flash memories can run executables without duplication, and this arrangement is called XIP.

However, current demand paging schemes do not utilize the XIP capability of the OneNAND flash. We will present a new demand paging technique for OneNAND flash that fully utilizes its XIP capability. This technique distinguishes pages that need to be loaded on to the on-chip SRAM for protracted use from pages that can be accessed directly from the SRAM buffers on the OneNAND chip and immediately discarded. This significantly reduces the number of pages loaded on to the on-chip SRAM, cutting the number of page faults, the page-fault penalty, and thus the energy consumption.

The contribution of this paper can be summarized as follows: this is the first implementation of demand paging based on the XIP capability of OneNAND flash. While we aim to optimize the choice of pages to load on to on-chip SRAM, the complexity of the optimal solution has led up to develop an algorithm that adopts a finite sliding window. This approach is based on accurate energy consumption models of memory components, and is intended to lead to a quality solution for commercial application. Averaged experimental results show that our method outperforms conventional LRU-based demand paging by 57% in terms of execution time and by 63% in terms of energy consumption. It is actually better than optimal off-line solution of the conventional demand paging problem, called MIN, with a saving of 30% in time and 40% of energy.

2. RELATED WORK

Flash memory is becoming more widely used in mobile embedded systems, to store code as well as data, due to its non-volatility, solid-state reliability and low power consumption. Flash memory is commonly categorized into two types: NOR and NAND. The NOR type is particularly well suited to code storage and execute-in-place (XIP) applications that require high-speed random access [6], while NAND flash requires sequential access and has a long access latency, making it suitable for data storage due to its lower cost per bit and higher density [6]. NAND flash memory is widely used in cellular phones and portable memory cards in consumer electronics.

Park et al. characterized the energy consumption of conventional paging mechanisms applied to embedded applications stored in NAND flash memory [2]. They proposed an energy-aware page replacement policy for the applications stored in NAND flash memory. Their approach requires virtual memory or hardware (MMU) support.

NAND XIP [1] allows direct code execution from NAND flash memory using a cache controller, but the additional cost of this hardware will be unacceptable in many low-end embedded systems.

In industry, NAND XIP [7][5] has been implemented using small buffers and I/O interface conversion. This new NAND architecture is commonly called Hybrid NAND flash. However, use of the XIP feature of Hybrid NAND has largely been limited to boot code execution.

Many authors [8][9][10][11] have dealt with the problem of data reuse in the design of a custom memory hierarchy. These approaches have used profile-based off-line analysis, with the target memory hierarchy and the data layout fixed at run-time. They are similar to our scheme in that the target memory system is multi-layered and the memory allocation policy plays an important role in performance and energy optimization. However, unlike previous authors, we change the memory hierarchy dynamically at run-time through judicious use of the OneNAND SRAM buffers and XIP. Depending on the access pattern of the referenced page, the OneNAND buffers may function as main memory, or as buffers that transfer data from secondary storage to main memory. Moreover, whereas previous work is focused on optimizing the cost of memory access, we try to optimize the page miss count, because the cost of loading missed pages is more significant than the memory access cost in NAND flash memory systems.

3. DEMAND PAGING FOR ONENAND FLASH MEMORIES

Demand paging is a virtual memory technique in which code or data is loaded from the secondary storage only when needed by a process. Traditional demand paging schemes are aimed at general-purpose systems which have hard disk drives as their secondary storage. However, increasing demand for memory has made demand paging a common feature of embedded systems, even though it generally requires the support of memory management unit (MMU). Many recent embedded processors, such as the Motorola 68060, Intel PXA255 and the ARM920T, support an MMU, which simplifies the inclusion of demand paging in embedded systems.

In addition to the performance issue, many portable embedded systems require high mobility. To cope with this property, recent embedded systems adopt new memory components such as flash memories which enable to reduce overall weight and power consumption, providing faster access to data than hard disks. Due to the introduction of new components into the storage hierarchy, it is necessary to re-evaluate demand paging as a method of fully exploiting the benefits of new types of memory.

We will first review the conventional paging system to understand its weakness in the context of new memory components such as OneNAND flash memory, and then we will briefly itemize the features of OneNAND flash memory from the paging perspective. Finally, we will summarize the performance criteria that must be considered in assessing a paging system for real-time embedded systems in more detail.

3.1 Conventional demand paging with OneNAND Flash

The standard paging scheme used with OneNAND flash memories is similar to the classical paging method used in general-purpose systems with hard disk drives. A OneNAND flash memory consists of an internal flash array and SRAM buffers. Fig. 2 depicts a typical demand paging procedure in a memory hierarchy that includes OneNAND flash memory. Because the OneNAND chip has two buffers, buffer-level interleaving can be used to hide some of the time taken to move data from flash array to the buffers. The

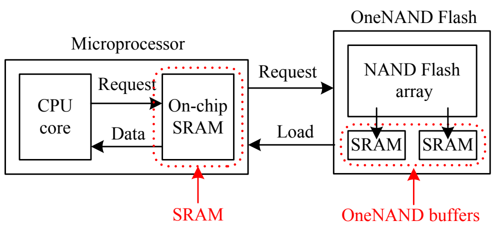


Figure 2: Conventional demand paging with a OneNAND flash memory.

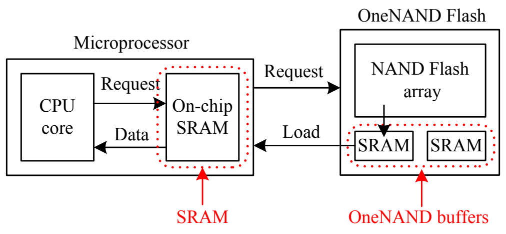
RAM between the CPU and the OneNAND flash memory can be either SRAM or DRAM. But SRAM is preferred in embedded systems because of its fast accessibility and low power, and we will assume the use of on-chip SRAM throughout this paper. To distinguish clearly between the on-chip SRAM and the SRAM buffers in the OneNAND flash memory, we will call them the ***SRAM*** and the ***OneNAND buffer*** respectively.

If a page fault occurs, the CPU asks the OneNAND flash memory to fetch the page. Once the request has been granted, the OneNAND flash memory first transfers the page from the flash array to the OneNAND buffers, and then the OneNAND buffers forward the page to the RAM. After this two-stage transfer is complete, the CPU resumes its execution by taking the appropriate data or instruction. The CPU can only access data residing on SRAM, so the number of accesses to a newly loaded page should ideally be sufficient to amortize the cost loading it from the OneNAND buffer to the on-chip SRAM. But there is no way to access any data in OneNAND flash memory without loading it to on-chip SRAM. Due to its similarity with a conventional hard disk drive, we can directly apply classical LRU (Least recently used) and FIFO methods to OneNAND flash memory systems. The theoretically optimal solution to this problem is incorporated in the MIN algorithm in [12].

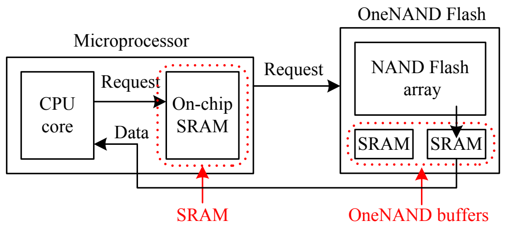
3.2 A new approach to demand paging using the OneNAND XIP feature

The main benefit of the XIP feature of OneNAND is that the CPU can treat the OneNAND buffer as a unit in a cache line, and access data directly. Using this feature, we can overcome the weakness of conventional demand paging method by avoiding loading data on to the on-chip SRAM if the expected number of accesses to a page will not be large enough to amortize its loading cost. This case is depicted in Fig. 3(b), where the CPU directly accesses the OneNAND buffer even though a page fault has occurred. Fig. 3(a) shows the page loading procedure which is similar to that shown in Fig. 2. The efficiency of this scheme depends critically on how the OneNAND buffers are used. There are three possible ways to use the two OneNAND buffers. The first way is to use them both for paging. This is the standard procedure that we have already mentioned and we will not analyze it further. The second technique is to use a single OneNAND buffer for paging while the other is dedicated to the support of direct accesses through XIP. In this case, we cannot exploit the buffer-level interleaving effect and page utilization will be low if a high-demand page is mapped to the buffer dedicated to XIP, and the benefit of XIP will then be lower than we expect. The third method is a hybrid of the previous two. A page is loaded into a OneNAND buffer from the flash array when a page fault occurs. This procedure is applicable to both methods but they are differentiated in the next step, in which we must determine whether to upload the data to the on-chip SRAM or to keep it in a OneNAND buffer. In the third method, we adaptively switch between these two actions so as to maximize page utilization.

Deciding whether the page in the OneNAND buffers should be uploaded to the on-chip SRAM for paging or retained for XIP



(a) Load-preferred page



(b) XIP-preferred page

Figure 3: Demand paging using the XIP feature of OneNAND flash memory.

access requires knowledge of the likely future access pattern of pages, and we will address this issue in detail in Section 4. The paging scheme can be further generalized to additional OneNAND buffers, but we limit our focus to the two OneNAND buffer which are available in contemporary OneNAND flash memories.

3.3 Performance criteria

Table 1 shows the performance of a paging system, in terms of energy consumption as well as performance, within a real-time embedded system. The memory access statistics are related to the number of data communications between the blocks of the system, as shown in Fig. 2. Obviously, the number of data communication in the system critically affects overall execution time. Inside a OneNAND flash memory, there are $N_{Flash2Buf}$ page transfers between flash array and the OneNAND buffers. Similarly, there are $N_{Buf2SRAM}$ transfers between the OneNAND buffers and the SRAM. These transfers are caused by page faults. We also need to measure the number of accesses to a unit of the cache line rather than to a page. N_{SRAM_Read} counts the accesses resulting from page hits, and N_{Buf_Read} is incremented when the CPU directly accesses the data residing on a OneNAND buffer. The four timing measures in Table 1 correspond to the four access counts that we have just itemized. Energy measures have not been considered in classical demand paging schemes designed for general purpose systems. Again the four energy parameters in Table 1 correspond to the access counts.

Using these 12 parameters, we can formulate the overall execution time and energy consumption of a memory system that incorporates a OneNAND flash memory as follows.

$$E_{Total} = N_{Flash2Buf} \cdot E_{Flash2Buf} + N_{Buf2SRAM} \cdot E_{Buf2SRAM} + N_{Buf\_Read} \cdot E_{Buf\_Read} + N_{SRAM\_Read} \cdot E_{SRAM\_Read}$$

$$T_{Total} = N_{Flash2Buf} \cdot T_{Flash2Buf} + N_{Buf2SRAM} \cdot T_{Buf2SRAM} + N_{Buf\_Read} \cdot T_{Buf\_Read} + N_{SRAM\_Read} \cdot T_{SRAM\_Read}$$

In conventional demand paging $N_{Flash2Buf}$ is equal to $N_{Buf2SRAM}$ and N_{Buf_Read} is zero. Using XIP these parameters depend on the buffer management policy.

Table 1: Performance parameters of the paging system

Memory access statistics	
$N_{Flash2Buf}$	Total number of page transfers from the flash array to the OneNAND buffer
$N_{Buf2SRAM}$	Total number of page transfers from the OneNAND buffer to the SRAM
N_{Buf_Read}	Total number of page reads from the OneNAND buffer due to cache line fetch
N_{SRAM_Read}	Total number of page reads from the SRAM due to cache line fetch
Timing parameters	
$T_{Flash2Buf}$	Time for a page transfer from the flash array to the OneNAND buffer
$T_{Buf2SRAM}$	Time for a page transfer from the OneNAND buffer to the SRAM
T_{Buf_Read}	Time for a page read from the OneNAND buffer
T_{SRAM_Read}	Time for a page read from the SRAM
Energy parameters	
$E_{Flash2Buf}$	Energy consumption for a page transfer from the flash array to the OneNAND buffer
$E_{Buf2SRAM}$	Energy consumption for a page transfer from the OneNAND buffer to the SRAM
E_{Buf_Read}	Energy consumption for a page read from the OneNAND buffer
E_{SRAM_Read}	Energy consumption for a page read from the SRAM

4. ONENAND BUFFER MANAGEMENT FOR DEMAND PAGING WITH XIP

We will now describe a buffer management technique called PM-XIP for the hybrid use of a OneNAND flash memory with two OneNAND buffers. The quality of a buffer management policy critically depends on its ability to predict future page access patterns. Obviously, the page access pattern cannot be statically formulated due to the unpredictable control flow of a particular piece of software. Chung et al. proposed a sliding window technique to predict the future behavior of service requests for dynamic power management (DPM) of hard disk drives [13]. Its effectiveness has been demonstrated [14] by comparing the technique with other adaptive DPM methods. Our problem is similar in the sense that we need an on-line prediction method based on a discrete history, and we use a similar method to predict which pages will be in demand in the future. We also need to predict the victim page, residing on the on-chip SRAM or on the OneNAND buffers which is in least demand and will be replaced by a new page from the flash array via the OneNAND buffer. We could use the same sliding window technique for victim page selection, but we use LRU instead because the victim page selection does not deal with a large number of pages compared to instruction codes.

Fig. 4 shows an example of a sliding window (which we will call a page history window) and shows how it is used to determine the page to be loaded to the on-chip SRAM. The size of the page history window is finite and it only stores a certain length of the access pattern: when a new page is accessed, the oldest information must be discarded. Pages are only replaced when a page fault occurs, and no changes are considered during a run of page hits. When a page fault occurs, the fault page is loaded into one of the OneNAND buffers (selected by the LRU policy) and then the pages on both the OneNAND buffers become candidates to be loaded to on-chip SRAM. If the previous access frequency of a page is more than a predefined threshold, it is loaded to the on-chip SRAM. If the previous access frequencies of both pages exceed the threshold, two victim pages must be selected from those residing on the on-chip SRAM. Fig. 5 shows pseudocode for PM-XIP.

The size of the page history window and the predefined threshold control the prediction quality. If the window is too large, the prediction is contaminated by old access patterns which may be quite different from recent patterns. But, if the window size is too small, there will insufficient information for a good prediction. A suitable window size can be determined by code profiling.

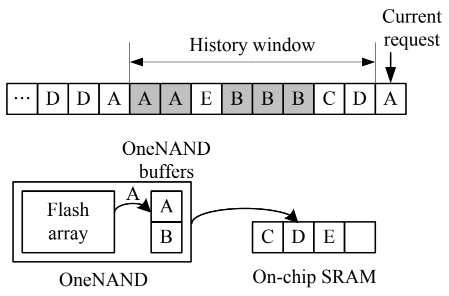


Figure 4: The page history window.

```
PM-XIP: A page manager for demand paging with OneNAND XIP
Input:  $\rho=(\rho_1, \rho_2, \cdots)$: page request sequence
        $\omega$: window size
        $\theta$: predefined threshold
Procedure:
   Page manager:
 1.  do
 2.     fetch next requested page $\rho_i$
 3.     if($\rho_i$ is available from the on-chip SRAM)
 4.        increase the SRAM page access count;
 5.     else if($\rho_i$ is available from the OneNAND buffer)
 6.        increase the OneNAND page access count;
 7.     else
 8.        call Page fault handler;
 9.     end if
10.     update the history window;
11.  until program is terminated
   Page fault handler:
12.  if(OneNAND buffer is full)
13.     discard one victim page from the OneNAND buffer in LRU order;
14.  end if
15.  load $\rho_i$ from the internal NAND array to the OneNAND buffer;
16.  increase the page load count;
17.  for each page in the OneNAND buffer
18.     if(the number of page occurrences in the history window
         exceeds $\theta$)
19.        if(SRAM is full)
20.           discard one victim page from the SRAM in LRU order;
21.        end if
22.        move current page from the OneNAND buffer
           to the on-chip SRAM;
23.        increase the page move count;
24.     end if
25.  end for
```

Figure 5: Pseudocode of OneNAND-XIP paging algorithm.

5. EXPERIMENTS

5.1 Experimental setup

We implemented a general-purpose paging system simulator that is capable of simulating various paging techniques including LRU, MIN and PM-XIP. Table 2 shows the benchmark applications, which are components of a UI or an image viewer. Page request sequences for input to the paging system simulator are captured by a system-level cycle-accurate simulator [15]. The simulated system has a 4-way associative 4KB instruction cache with a block size of 4-words. The logical page size is set to 1KB, which is the same as the OneNAND buffer size. Page request sequences are captured until the application terminates, except the page request sequence for FFT was truncated at 1M due to its long execution time. We performed the simulation with a 4KB, 8KB, 16KB or a 32KB on-chip SRAM depending of the size of the benchmark

Table 2: Benchmark applications

Application	Code size (KB)	Number of page requests
DJPEG	148	295081
CJPEG	195	268980
Basicmath	32	1889410
FFT	30	1000000

Table 3: Timing and energy costs for the simulation

Time cost	Value (μs)	Energy cost	Value (nJ)
$T_{Flash2Buf}$	29.33	$E_{Flash2Buf}$	1295.48
$T_{Buf2SRAM}$	12.86	$E_{Buf2SRAM}$	1056.21
T_{Buf_Read}	0.22	E_{Buf_Read}	15.24
T_{SRAM_Read}	0.04	E_{SRAM_Read}	1.79

codes. Here the size of the SRAM refers to the dedicated page buffer and not to the entire on-chip SRAM.

The timing and energy models of the on-chip SRAM are obtained from data sheets [16]. We chose a 64KB high-density single-port synchronous static RAM for the on-chip SRAM in this simulation. We also used Samsung's KFG5616X1A, 256Mb OneNAND flash, and characterized the timing and energy models from previous authors' work [17] because the information in the data sheets [4] sufficient to calculate all the timing and energy parameters used in the simulation. The clock frequencies of the on-chip SRAM and the OneNAND flash were set to 100MHz and 50MHz respectively, which are values commonly used for contemporary embedded systems. The parameters for the timing and energy model are shown in Table 3.

5.2 Simulation results

We have explored our demand paging method by varying three key parameters: the window size and threshold value, which determine the quality of prediction by the algorithm, while the on-chip SRAM size characterizes the architecture. To assess the impact of these parameters, we performed the simulation for window sizes from 2 to 1024, threshold values between 0% and 100% of the window size, and on-chip SRAMs of 4KB and 32 KB.

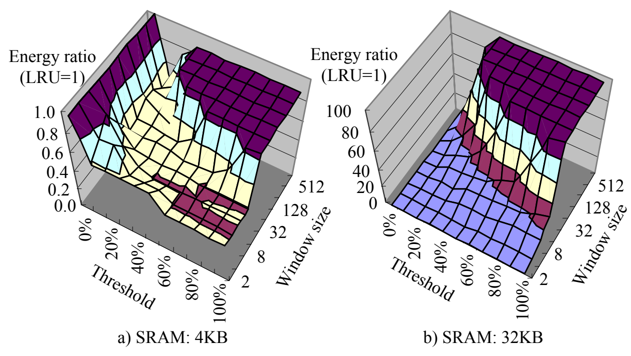


Figure 6: Energy consumption of PM-XIP (DJPEG).

Fig. 6 and Fig. 7 show the energy consumption and the elapsed time of PM-XIP, normalized to that of LRU, when running DJPEG on 4KB and 32KB SRAM. With a 4KB SRAM, the best results in energy consumption are obtained when the window size is 8 and the threshold value is 50% of the window size, as shown in Fig. 6(a). The best results in elapsed time are obtained where the window size and the threshold value are same to that of energy consumption. Energy consumption is 61% better than the standard method of demand paging, and the elapsed time is reduced by 57%.

With a 32KB SRAM, the energy consumption and the elapsed time are best when both parameters are 0, as shown in Fig. 6(b) and Fig. 7(b), meaning that it is always best to move a page to the

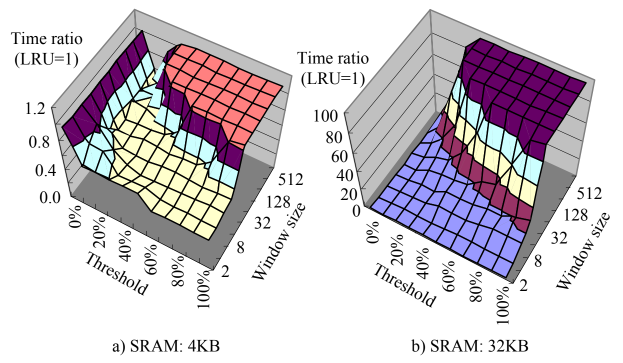


Figure 7: Elapsed time for PM-XIP (DJPEG).

Table 4: Time (ms) and energy (mJ) results for PM-XIP running on SRAMs of 4KB and 32KB (DJPEG, ω=8, θ=4)

Performance index	SRAM size = 4KB			SRAM size = 32KB		
	PM-XIP	LRU	MIN	PM-XIP	LRU	MIN
$N_{Flash2Buf}$	16968	33773	22266	3523	100	85
$N_{Buf2SRAM}$	6115	33773	22266	64	100	85
N_{Buf_Read}	161787	0	0	38875	0	0
N_{SRAM_Read}	133294	295081	295081	256206	295081	295081
$(E \cdot N)_{Flash2Buf}$	21.98	43.75	28.85	4.56	0.13	0.11
$(E \cdot N)_{Buf2SRAM}$	6.46	35.67	23.52	0.07	0.11	0.09
$(E \cdot N)_{Buf\_Read}$	2.47	0	0	0.59	0	0
$(E \cdot N)_{SRAM\_Read}$	0.24	0.53	0.53	0.46	0.53	0.2
E_{Total}	31.15	79.95	52.9	5.68	0.77	0.73
$(T \cdot N)_{Flash2Buf}$	497.67	990.56	653.06	103.33	2.93	2.49
$(T \cdot N)_{Buf2SRAM}$	78.64	434.32	286.34	0.82	1.29	1.09
$(T \cdot N)_{Buf\_Read}$	35.59	0	0	8.55	0	0
$(T \cdot N)_{SRAM\_Read}$	5.33	11.8	11.8	10.25	11.8	11.8
T_{Total}	617.23	1436.69	951.21	122.95	16.02	15.39

on-chip SRAM when it is accessed from the OneNAND buffers. This occurs because the whole working set can fit on to the on-chip SRAM. This is unlikely in practice because semiconductor area is a critical design constraint for cost-effectiveness.

Table 4 shows more details of the simulation results at the optimal point for a 4KB SRAM, when the window size is 8 and the threshold is 4. The number of page transfers from the buffer to the on-chip SRAM ($N_{Buf2SRAM}$) is significantly reduced by 64.0%, because PM-XIP is transferring popular pages to on-chip SRAM whereas LRU and MIN transfer all pages. Our method also reduces the number of page transfers from the flash array to the buffer ($N_{Flash2Buf}$), which reduces the number of page faults compared to LRU by 49.8% and by 23.8% compared to MIN, with a 4KB SRAM.

With a large 32KB SRAM in Table 4, both $N_{Flash2Buf}$ and $N_{Buf2SRAM}$ are dramatically reduced. However, $N_{Flash2Buf}$ is much larger for our method than that for the other two. This is due to the aggressive replacement policy results from a small page history window. When the area constraint is not tight and there is a large on-chip SRAM, we can take a more conservative approach by choosing an arbitrary window size and setting the threshold to zero, which produces the same results as LRU. This shows how we can modify the aggressiveness of PM-XIP without changing any hardware.

We conducted another set of simulations to understand the performance variation when we have an on-chip SRAM space constraint. Most real-time applications pin their kernel code on the on-chip SRAM and other code has to share the remaining space. It is not easy to estimate how large this will be at an early design stage, since it very much depends on the code optimization policy. For this reason, a demand paging method should be as insensitive as possible to the on-chip SRAM space, and require few changes

Table 5: Performance and energy comparison of PM-XIP(8,4) and PM-XIP$(\omega_{opt}, \theta_{opt})$ (normalized to LRU)

SRAM size (KB)	Application	T_{total}			E_{total}		
		PM-XIP (8,4)	PM-XIP $(\omega_{opt}, \theta_{opt})$	MIN	PM-XIP (8,4)	PM-XIP $(\omega_{opt}, \theta_{opt})$	MIN
4	DJPEG	0.43	0.43	0.66	0.39	0.38	0.66
	CJPEG	0.44	0.40	0.63	0.40	0.36	0.63
	Basicmath	0.45	0.34	0.59	0.40	0.28	0.59
	FFT	0.61	0.48	0.66	0.54	0.39	0.66
8	DJPEG	0.43	0.41	0.64	0.36	0.35	0.63
	CJPEG	0.38	0.38	0.65	0.33	0.32	0.65
	Basicmath	0.47	0.42	0.55	0.40	0.35	0.54
	FFT	0.75	0.57	0.65	0.64	0.47	0.65
16	DJPEG	2.11	0.89	0.76	1.87	0.80	0.75
	CJPEG	3.23	1.00	0.92	2.83	1.00	0.92
	Basicmath	0.46	0.36	0.44	0.38	0.30	0.43
	FFT	0.05	0.04	0.21	0.05	0.04	0.20
32	DJPEG	7.67	1.00	0.96	7.45	1.00	0.95
	CJPEG	6.66	1.00	0.95	6.51	1.00	0.94
	Basicmath	5.71	1.00	1.00	5.77	1.00	1.00
	FFT	1.41	1.00	1.00	1.66	1.00	1.00

to hardware or software to cope with different configurations. To assess the quality of our method from this perspective, we performed simulations while varying the space remaining on the on-chip SRAM. We selected two cases and compared PM-XIP with LRU and MIN: a window of 8 and a threshold of 4, and a window of ω_{opt} and a threshold of θ_{opt}, where ω_{opt} and θ_{opt} are optimized for least energy and best performance. We also consider different sizes of instruction codes to see the interplay between the remaining SRAM space and the code size. The simulation results are summarized in Table 5.

We have already shown that PM-XIP is comparable to LRU and MIN when the remaining SRAM space is large enough, and that our method outperforms the others when the remaining SRAM space is small. The interesting point in this table is the comparison between the two conditions. With 4KB or 8KB of SRAM environments the optimal parameters slightly outperform 8 and 4 when running DJPEG and CJPEG. However, the optimal values perform much better on Basicmatch and FFT, in terms of energy consumption as well as elapsed time. This is closely related to the size of working set (or the number of pages actively demanded). Only 17 pages are actively accessed by the FFT code, while more than 80 pages are actively accessed by DJPEG code. The page size in our experiment is 1KB, so there can be only 4 pages in a 4KB SRAM and 8 pages in an 8KB SRAM environment. This suggests that our method can be effectively tuned for a certain range of page sizes and proportions of remaining space in the SRAM. Parameter optimization improves the performance by 10%, even when the working set is 17.5 times larger than the SRAM size, which is the case for CJPEG and a 4KB SRAM. The benefit of optimization becomes clearer when we consider 16KB and 32KB of SRAM. In these environments, PM-XIP performs significantly better with optimal parameters due to the reduction of page size and the amount of remaining SRAM.

Of course there are storage and computation overheads in maintaining the page history window. However, the size overhead is negligible for all the benchmarks because the optimal or near-optimal points always exist for a window size of less than 16. The computational overhead consists of the threshold value calculation and updating of the page history window. The threshold value is only calculated once for each page fault, and the load is negligible. The page history window must be updated for every page request, but implemented using just one modulus operation for indexing and one write operation for recording the current page request. Moreover, accesses to the page history window are highly likely to be performed in the CPU cache because these accesses occurs very frequently and the window is small.

6. CONCLUSION

We have introduced an innovative demand paging scheme called PM-XIP, that fully utilizes the XIP capability of OneNAND flash. PM-XIP can deal with large size non-real-time tasks without increasing the on-chip SRAM footprint, which is crucial in cost-effective embedded systems. After pinning the real-time tasks in SRAM, modern embedded systems can suffer lack of memory to run other tasks, such as a quality user interface and DRM. PM-XIP runs 57% faster and requires 63% less energy, on average, when compared to the conventional LRU-based demand paging technique. PM-XIP even outperforms the optimal off-line demand paging technique, called MIN, by as much as 30% in terms of execution time and by 40% in terms of energy consumption. PM-XIP enables aggressive reduction of the on-chip SRAM footprint, even below the minimum working set size. We expect that the combination of OneNAND flash with a novel demand paging scheme such as PM-XIP will provide a significantly enhanced demand paging performance compared to conventional NAND flash.

7. REFERENCES

[1] C. Park, J. Seo, S. Bae, H. Kim, S. Kim, and B. Kim, "A low-cost memory architecture with nand xip for mobile embedded systems," in *Proceedings of the 1st IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, pp. 138 – 143, October 2003.

[2] C. Park, J.-U. Kang, S.-Y. Park, and J.-S. Kim, "Energy-aware demand paging on nand flash-based embedded storages," in *Proceedings of the 2004 International Symposium on Low Power Electronics and Design*, pp. 338 – 343, August 2004.

[3] C. Park, J. Lim, K. Kwon, J. Lee, and S. L. Min, "Compiler-assisted demand paging for embedded systems with flash memory," in *Proceedings of the 4th ACM International Conference on Embedded Software (EMSOFT'04)*, pp. 114 – 124, September 2004.

[4] *KFG5616x1A x16 OneNAND Specification*. Samsung Electronics, Co. Ltd, http://www.samsungelectronics.com, December 2005.

[5] *OneNAND Features & Performance*. http://www.samsung.com/Products/Semiconductor/Memory /appnote/onenand_features_performance_051104.pdf: Samsung Electronics, 2005.

[6] *Two Technologies Compared: NOR vs. NAND*. White Paper, 91-SR-012-04-8L, Rev 1.1: M-Systems, 2003.

[7] *M-Systems DOC H3*. http://www.m-systems.com/NR/rdonlyres/C7C40791-65D7-4807-B857-E0E8D28E8EAE/0/DOC_H3_DS_Rev01.pdf: M-Systems, 2006.

[8] P. R. Panda, N. D. Dutt, and A. Nicolau, "Efficient utilization of scratch-pad memory in embedded processor applications," in *Proceedings of the 1st European Design and Test Conference*, pp. 7 – 11, March 1997.

[9] J. P. Diguet, S. Wuytack, F. Catthoor, and H. D. Man, "Formalized methodology for data reuse exploration in hierarchical memory mappings," in *Proceedings of the IEEE International Symposium on Low Power Design*, pp. 30 – 35, August 1997.

[10] E. Brockmeyer, M. Miranda, H. Corporaal, and F. Catthoor, "Layer assignment techniques for low energy in multi-layered memory organisations," in *Proceedings of the 6th ACM/IEEE Design and Test in Europe Conference*, pp. 1070 – 1075, March 2003.

[11] I. Issenin, E. Brockmeyer, M. Miranda, and N. Dutt, "Data reuse analysis technique for software-controlled memory hierarchies," in *Proceedings of the 7th ACM/IEEE Design and Test in Europe Conference*, pp. 202 – 207, Feb 2004.

[12] L. A. Belady, "A study of replacement algorithms for virtual storage computers," *IBM Systems Journal*, vol. 5, no. 2, pp. 78–101, 1966.

[13] E.-Y. Chung, L. Benini, A. Bogliolo, Y.-H. Lu, and G. D. Micheli, "Dynamic power management for nonstationary service requests," *IEEE Transactions on Computers*, vol. 51, no. 11, pp. 1345–1361, 2002.

[14] Y.-H. Lu and G. D. Micheli, "Comparing system-level power management policies," *IEEE Design and Test of Computers*, vol. 18, no. 2, pp. 10–19, 2001.

[15] I. Lee, Y. Choi, Y. Cho, Y. Joo, H. Lim, H. G. Lee, H. Shim, and N. Chang, "Web-based energy exploration tool for embedded systems," *IEEE Design and Test of Computers*, vol. 21, no. 6, pp. 572 – 586, 2004.

[16] *STD130(Rev. 2.1) 0.18μm 1.8V CMOS Standard Cell Library for Pure Logic Products*. Samsung Electronics, Co. Ltd, http://www.samsungelectronics.com, February 2004.

[17] *SECM: SNU Energy Characterizer for Memory Devices, http://elpl.snu.ac.kr/measurement/sec.htm*. 2005.