가상 I/O 세그먼트를 이용한 OneNAND 플래시 메모리의 읽기 성능 향상 기법

(Improving the Read Performance of OneNAND Flash Memory using Virtual I/O Segment)

현 승 환 † 고 건 ††
(Hyun Seunghwan) (Koh Kern)

요 약 OneNAND 플래시는 NAND 플래시와 NOR 플래시의 장점을 모두 가진 고성능 하이브리드 플래시 메모리이다. OneNAND 플래시는 NAND 플래시의 장점들을 그대로 가지고 있을 뿐 아니라, 그동안 NAND 플래시의 단점으로 지적되던 느린 읽기 성능을 획기적으로 개선하였다. 그 결과 OneNAND 플래시는 휴대폰 및 디지털 카메라, PMP, 휴대용 게임기와 같은 고성능 휴대용 정보기기를 위한 최적의 스토리지 솔루션으로 각광받고 있다. 하지만 Linux를 비롯하여 현재 사용되고 있는 대부분의 범용 운영체제들은 가상 메모리와 블록 I/O 계층 구조의 제약으로 인해 OneNAND 플래시의 뛰어난 읽기 성능을 제대로 활용하지 못하는 문제를 안고 있다. 이에 본 연구에서는 기존의 소프트웨어 계층 구조 하에서 OneNAND 플래시의 읽기 성능을 최대한 활용하기 위한 기법인 가상 I/O 세그먼트의 활용을 제안한다. 실제 구현을 통한 실험 결과는 제안된 기법이 OneNAND 플래시의 읽기 수행 시간을 기존에 비해 최고 54%까지 단축할 수 있음을 증명하였다.

키워드 : OneNAND 플래시 메모리, 이중 버퍼링, Linux, MTD 서브시스템, 가상 I/O 세그먼트

Abstract OneNAND flash is a high-performance hybrid flash memory that combines the advantages of both NAND flash and NOR flash. OneNAND flash has not only all virtues of NAND flash but also greatly enhanced read performance which is considered as a downside of NAND flash. As a result, it is widely used in mobile applications such as mobile phones, digital cameras, PMP, and portable game players. However, most of the general purpose operating systems, such as Linux, can not exploit the read performance of OneNAND flash because of the restrictions imposed by their virtual memory system and block I/O architecture. In order to solve that problem, we suggest a new approach called virtual I/O segment. By using virtual I/O segment, the superior read performance of OneNAND flash can be exploited without modifying the existing block I/O architecture and MTD subsystem. Experiments by implementations show that this approach can reduce read latency of OneNAND flash as much as 54%.

Key words : OneNAND flash memory, dual buffering, Linux, MTD subsystem, virtual I/O segment

1. 서 론

NAND 플래시는 저렴한 가격과 대용량, 빠른 쓰기 속도 등의 많은 장점에도 불구하고, NOR 플래시에 비해 느린 접근 속도 때문에 주로 사용자 데이타의 저장을 위한 용도로만 왔다. OneNAND 플래시는 NAND 플래시의 이러한 단점을 획기적으로 개선한 NAND 기반 하이브리드 플래시(hybrid flash) 메모리로써, 빠른 쓰기 속도와 높은 가격 효율성과 같은 기존 NAND 플래시의 장점 외에 향상된 읽기 속도와 보다 높은 신뢰성, 시스템 부팅이 가능하며 제한적인 XIP(eXecute-In-Place) 기능 등의 여러 가지 장점을 가지고 있다[1].

† 학생회원 : 서울대학교 컴퓨터공학부
kakjagi@oslab.snu.ac.kr
†† 종신회원 : 서울대학교 컴퓨터공학부 교수
kernkoh@oslab.snu.ac.kr
논문접수 : 2007년 12월 17일
심사완료 : 2008년 9월 4일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제14권 제7호(2008.10)

그 중에서도 OneNAND 플래시가 주목 받는 가장 큰 이유는 빠른 읽기 성능 때문이다. 휴대폰으로 대표되는 오늘날의 모바일 디바이스들이 불과 수년 전에는 상상할 수 없었던 다양한 기능을 내장하게 되면서, 가격 및 확장성 측면에서 큰 장점을 가진 NAND 플래시 기반의 요구 페이징 아키텍처가 고성능, 다기능 모바일 디바이스를 위한 새로운 메모리 아키텍처로 사용되기 시작했다[2,3]. 하지만 NAND 플래시 기반 요구 페이징 시스템은 성능 문제로 인해 고성능, 실시간성이 요구되는 응용에서는 사용하기 어려운 단점도 가지고 있다. 요구 페이징 시스템의 성능은 저장장치의 읽기 성능에 크게 좌우되는데 NAND 플래시의 읽기 성능은 모바일 디바이스가 요구하는 고성능 및 실시간성을 만족시키기에 부족하기 때문이다.

OneNAND 플래시는 이러한 상황에서 NAND 플래시를 대체할 수 있는 가장 매력적인 대안이다. 표 1에서 볼 수 있듯이 OneNAND 플래시는 읽기, 쓰기 성능에서 모두 기존의 NAND 플래시에 비해 우수하며, 특히 최고 읽기 속도에서는 NOR 플래시에도 근접할 만큼 뛰어난 성능을 보여준다. 또한 NAND 플래시 수준의 낮은 가격과 큰 저장용량, 그리고 보다 높은 신뢰성을 제공하는 저장장치로써 오늘날의 모바일 디바이스가 요구하는 모든 요구 사항을 하나의 칩으로써 만족시켜준다. 때문에 OneNAND 플래시는 고성능, 다기능 모바일 디바이스를 위한 최적의 저장장치로 평가 받고 있다.

하지만 현재 사용되고 있는 대부분의 범용 운영체제들은 가상 메모리 구조 및 I/O 계층 구조의 제약으로 인해 OneNAND 플래시의 뛰어난 읽기 성능을 충분히 활용하지 못하는 문제를 안고 있다. 이는 현재의 I/O 계층 구조가 디스크와 같은 블록 디바이스 및 기존의 NOR, NAND플래시 메모리들만을 고려하여 설계되어 있어서 OneNAND 플래시의 특성을 충분히 활용하지 못하기 때문이다. 이에 본 연구에서는 이러한 문제의 요인과 해결 방안에 대해 연구하였으며, 그 결과로 가상 I/O 세그먼트 기법의 사용을 제안한다.

본 논문은 두 가지 면에서 그 의의를 지닌다. 첫번째는 쓰기 성능에 집중된 기존의 연구와는 달리 읽기 성능의 향상을 위한 연구라는 점이다. 플래시 메모리, 특히 NAND 플래시 메모리는 일반 메모리 소자와는 달리 쓰기 및 신뢰성에 관련된 까다로운 특성을 가지고 있다. 때문에 NAND 플래시 메모리에 대한 기존의 연구들은 이러한 물리적 특성을 극복하기 위한 기법들에 집중되어 왔다. 블록 맵핑(block mapping) 기법, 삭제 단위 회수(erase unit reclamation) 및 마모 균등화(wear-leveling) 기법, 배드 블록 관리 기법, 오류 발견 및 정정(error detection and correction) 기법들이 그 대표적인 연구 분야[4,5]이다. 최근에는 NAND 플래시 기반의 요구 페이징 시스템에 대한 연구도 많이 이루어지고 있다. 요구 페이징 시스템에서 가장 중요한 분야인 페이지 회수(page reclama tion) 기법 분야에서CFLRU[6], LIRS-WSR[7], CFLRU/C,E, DL-CFLRU/E[8] 등이 제안되었으며 플래시 기반의 스왑 장치 관리 성능을 향상시키기 위한 기법[9,10]들이 제안된 바 있다. 그러나 앞서 설명했듯이 이 연구들은 NAND 플래시의 읽기 성능보다는 쓰기 특성으로 인한 성능과 에너지 문제를 해결하는데 집중하였으며 읽기 성능의 향상에는 큰 의미를 두지 않았다. 두번째는 본 연구가 범용 운영체제에서 OneNAND 플래시의 읽기 성능을 최대한 활용하기 위한 실질적인 방법을 제시했다는 점이다. 물론 OneNAND 플래시의 읽기 성능과 특성을 활용하고자 하는 시도는 이미 존재한다. OneNAND 플래시의 제한적인 XIP 기능과 읽기 성능을 활용하여 요구 페이징 시스템의 페이지 반입 비용을 최소화하는 기법[11,12] 발표되었고, OneNAND 플래시의 데이타 적재(load) 작업과 페이지 할당 작업을 병렬적으로 수행하여 페이지 폴트 시의 처리 시간을 줄여주는 기법[3]이 발표된 바 있다. 하지만 이 연구들은 모두 극도로 적은 양의 자원을 이용하는 내장형 운영체제 및 실시간 운영체제에서의 활용을 목표로 한 것으로, 오늘날의 고성능, 다기능 모바일 디바이스에서 사용되는 범용 운영체제에 적용하기에는 힘든 연구라는 한계를 가지고 있다.

본 논문의 이후의 구성은 다음과 같다. 2장에서는 OneNAND 플래시의 특성과 현재의 I/O 계층구조에서 사용시의 문제점을 설명하였고, 3장에서 이를 해결하기 위한 가상 I/O 세그먼트의 개념 및 최적화 기법에 대해서 소개하였다. 이어지는 4장에서는 실험을 통해 이 기법들의 성능과 효과를 분석하였고 5장에서 연구 결과를 요약하였다.

2. OneNAND 플래시 메모리

2.1 OneNAND 플래시와 이중 버퍼링 기법

OneNAND 플래시는 NAND 플래시 기반의 하이브

표 1 플래시 메모리 성능 비교

	OneNAND 1Gb(A), (83Mhz)	NAND 4Gb (M, SLC)	NOR (SLC, 66Mhz)
Read Performance	108MB/s	28.5MB/s	133MB/s
Write Performance	9.3MB/s	8MB/s	0.17MB/s

리드 플래시 메모리이다[1,13]. 그림 1에서 볼 수 있듯이 OneNAND 플래시는 데이타를 저장하는 NAND 플래시 코어 외에 복수의 버퍼램(BufferRAM), 추가적인 제어 회로(control logic) 및 에러 수정 회로(error correction logic), 고속의 NOR 호스트 인터페이스(host interface)를 포함하고 있어 기존의 NAND 플래시에 비해 다양한 기능과 향상된 성능을 제공한다.

OneNAND 플래시의 여러 특징 중 가장 주목할만한 것은 NAND 플래시에 비해 획기적으로 개선된 I/O 성능이다. OneNAND 플래시가 NAND 플래시 기반임에도 불구하고 그보다 뛰어난 I/O 성능을 발휘할 수 있는 가장 큰 이유는 두 개의 데이타램(DataRAM)을 이용한 이중 버퍼링(dual buffering) 기법 때문이다. OneNAND 플래시로부터 데이타를 읽는 작업은 NAND 페이지를 경유 버퍼(stopover buffer)에 저장하는 적재(load) 작업과 이를 호스트 메모리로 전송하는 읽기(read) 작업의 두 단계로 이루어진다. OneNAND 플래시에서는 두 개의 데이타램을 번갈아 경유 버퍼로 이용해서 적재와 읽기 작업을 병렬적으로 수행하는 것이 가능하며, 이렇게 적재와 읽기, 혹은 반대로 쓰기(write)와 프로그램(program)을 동시에 수행하는 기법을 이중 버퍼링이라고 한다[1,13].

이중 버퍼링 기법은 다수의 페이지에 대한 읽기와 쓰기 속도를 기존 NAND 플래시에 비해서 획기적으로 개선해준다. 그림 2는 이중 버퍼링을 통한 읽기 성능 향상의 예로써, 총 4개의 NAND 페이지를 읽을 때 적재와 읽기를 순차적으로 수행한 경우(그림의 위)와 병렬적으로 수행한 경우(그림의 아래)의 수행 시간 차이를 보여준다. 그림으로부터 적재와 읽기를 병렬적으로 수행했을 때 총 3개의 페이지를 읽거나 적재하는 시간만큼의 이득을 본다는 것을 알 수 있다. 이렇듯 이중 버퍼링 기법은 OneNAND 플래시의 읽기 성능을 활용하는데 있어서 가장 중요한 기법이다.

2.2 Linux 운영체제에서 OneNAND 플래시 성능 활용의 문제점

Linux를 비롯한 현재의 범용 운영체제에서 OneNAND 플래시의 읽기 성능을 제대로 활용하지 못하는 것은 현재의 I/O 계층 구조의 제약으로 인해 OneNAND 플래시의 이중 버퍼링 기법의 사용이 제한되기 때문이다. 이중 버퍼링을 사용하기 위해서는 OneNAND 플래시를 제어하는 저수준 드라이버에 충분한 크기의 요청이 전달되어야 하는데, 현재 Linux 운영체제의 I/O 계층 구조에서는 이 같은 조건 충족이 불가능하다.

Linux 시스템에서 플래시 메모리에 대한 I/O는 범용 블록 계층(generic block layer)과 플래시 기반 블록 디바이스(flash-based block device), MTD 서브시스템(memory technology device subsystem)[14]을 통해서 이루어진다. 그림 3에 이러한 계층 구조가 기술되어 있

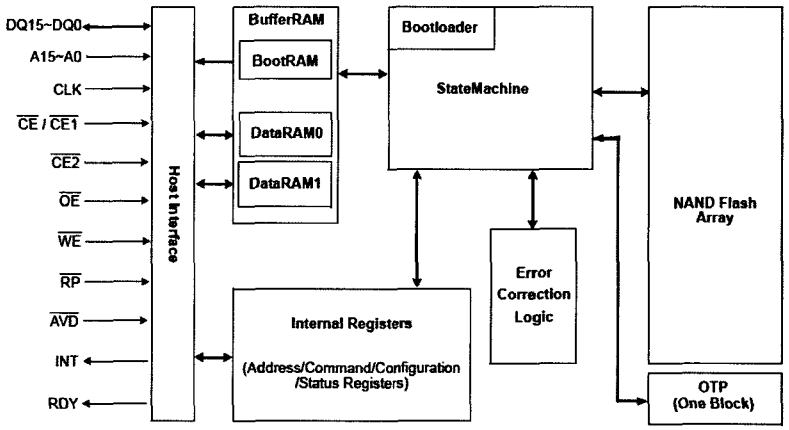

그림 1 OneNAND 플래시 메모리의 구조[1]

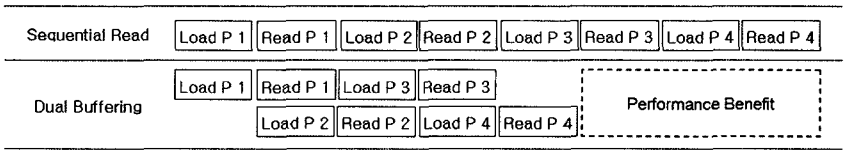

그림 2 읽기 이중 버퍼링의 효과

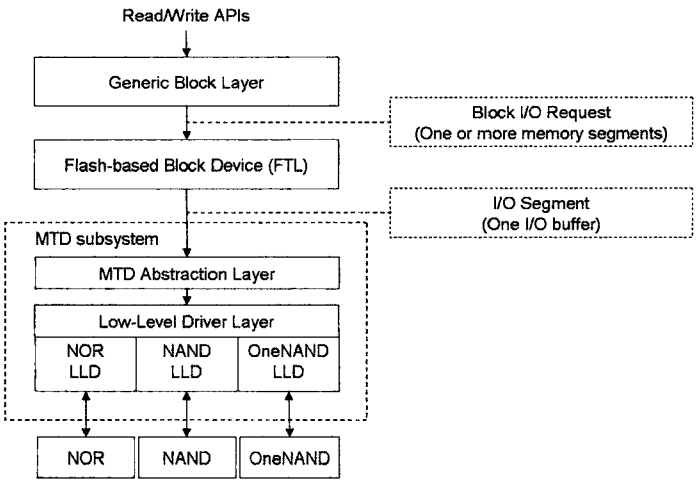

그림 3 Linux 범용 블록 계층과 MTD 서브시스템의 계층 구조

다. 최상위의 범용 블록 계층은 다양한 블록 디바이스에 대해 공통적인 인터페이스를 제공하는 커널 계층으로, 블록 디바이스에 대한 I/O 요청을 생성, 병합, 스케줄한 후 이를 특정 디바이스 드라이버에 전달하는 역할을 한다. 플래시 기반 블록 디바이스는 전달된 블록 I/O 요청을 해석한 후 MTD 서브시스템이 이해할 수 있는 형태로 번역(translation)하여 전달한다. MTD 서브시스템은 NOR와 NAND, OneNAND 플래시와 같은 다양한 종류의 메모리 디바이스들에 대한 공통 인터페이스를 제공하는 드라이버 계층으로써, 저수준 드라이버(low-level driver)를 통해 물리적인 I/O를 처리한다.

OneNAND 플래시에 대한 I/O는 이러한 계층 구조 안에서 다음과 같은 단계를 거쳐 이루어진다. 먼저 커널이 I/O를 요청하고 해당 요청이 파일 시스템을 거쳐 범용 블록 계층에 전달된다. 범용 블록 계층은 이를 I/O 요청 구조체로 만든 후 I/O 요청 큐에서의 스케줄링을 거쳐 플래시 기반 블록 디바이스에 전달한다. 이때 플래시 기반 블록 디바이스에 전달되는 I/O 요청 구조체는 그림 4와 같이 다수의 연속된 디바이스 블록과 다수의 비연속적인 메모리 버퍼에 대한 사상(mapping)으로 표현되는데, 이들 각각의 메모리 버퍼를 I/O 세그먼트라고 한다[15]. 플래시 기반 블록 디바이스는 전달된 I/O 요청을 I/O 세그먼트 단위로 분해한 후, MTD 서브시스템의 저수준 드라이버를 호출하여 I/O를 처리한다.

여기서의 문제는 MTD 서브시스템의 I/O 처리 단위가 I/O 세그먼트라는 것이다. 일반적으로 I/O 세그먼트의 최대 크기는 가상 메모리의 페이지 크기와 동일하므로[15], 이는 곧 저수준 드라이버에서 한번에 처리하는 I/O 요청의 최대 크기가 페이지의 크기인 4KB에 한정

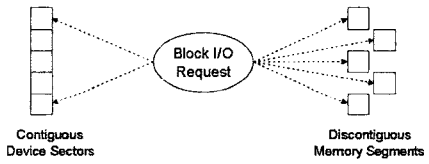

그림 4 블록 I/O 요청 구조체

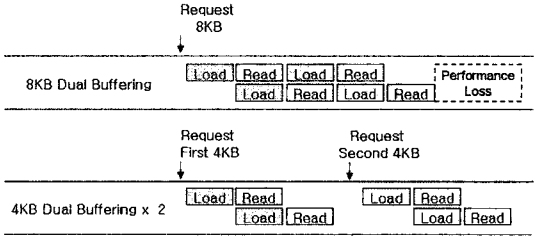

그림 5 세그먼트 크기 제한으로 인한 성능 손실의 예

된다는 것을 뜻한다. 이렇게 작은 단위로 I/O를 처리할 경우 이중 버퍼링을 충분히 활용하지 못할 뿐만 아니라 저수준 드라이버의 반복된 호출에 따른 불필요한 비용이 발생하여 결과적으로 I/O 성능이 떨어지게 된다. 그림 5는 2KB 크기의 페이지를 가진 OneNAND 플래시에서 총 8KB의 읽기 요청을 처리하는 예를 통해 이러한 문제점을 설명해준다. 만일 8KB의 요청을 한번에 처리한다면 총 3번의 이중 버퍼링을 이용하여 읽기 작업을 수행할 수 있지만, 이를 4KB I/O 세그먼트 단위로 처리한 경우에는 이중 버퍼링을 2번밖에 활용할 수 없을 뿐만 아니라 함수 호출에 관련된 부가적인 비용이 발생하여 전체 읽기 요청의 완료 시간이 그만큼 지연되게 된다.

이상의 논의는 현재 Linux 운영체제에서 OneNAND 플래시의 읽기 성능을 활용하지 못하는 요인을 설명해 준다. 우선 가상 메모리 및 I/O 계층 구조의 문제로 인해 MTD 서브시스템에 전달되는 읽기 버퍼의 크기가 I/O 세그먼트의 크기인 4KB로 제한되고, 이 때문에 OneNAND 플래시의 이중 버퍼링 기법을 충분히 활용하지 못하게 되는 것이다.

3. OneNAND 플래시의 읽기 성능 향상 기법

3.1 가상 I/O 세그먼트

I/O 세그먼트의 크기 제약으로 인해 이중 버퍼링을 활용하지 못하는 문제는 읽기 요청의 크기에 맞게 I/O 세그먼트의 크기를 확장함으로써 해결할 수 있다. 가상 I/O 세그먼트 기법은 읽기 요청을 구성하는 다수의 I/O 세그먼트들을 가상 주소 상에서 하나로 병합함으로써 이를 실현하는 기법이다(그림 6). 그림 7은 가상 주소의 조작을 통해 가상 I/O 세그먼트를 만드는 방법을 보여준다. 가상 메모리를 사용하는 운영체제에서는 하나의 물리 메모리 영역을 다수의 가상 주소에 맵핑할 수 있는데, 이를 이용하면 다수의 비연속적인 페이지 영역을 하나의 연속적인 주소 공간에 맵핑하는 것이 가능하다. 그림에서는 가상 주소 공간에서 비연속적으로 존재하는 4개의 페이지를 임의의 연속적인 16KB의 가상 주소 공간에 맵핑한 예를 보여준다. 새로 맵핑된 주소 공간은 16KB의 연속된 메모리 영역처럼 사용 가능하며, 이 주소 영역을 통한 메모리 접근은 원래의 비연속적인 주소를 통한 메모리 접근과 완전히 동일한 효과를 가진다. 이 때 이렇게 생성된 가상 주소 상의 메모리 영역을 가상 I/O 세그먼트라고 한다.

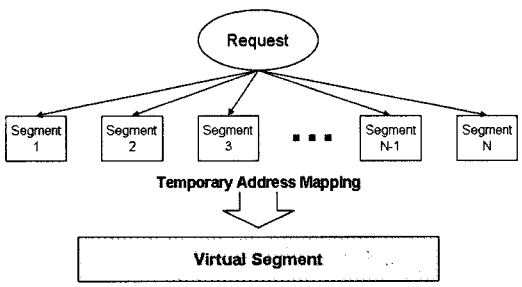

그림 6 가상 I/O 세그먼트의 개념

가상 I/O 세그먼트를 사용하면 I/O 요청에 속하는 전체 I/O 세그먼트를 한번에 저수준 드라이버까지 전달할 수 있으며, 이는 저수준 드라이버 단에서 이중 버퍼링 기법을 효과적으로 사용할 수 있게 해준다. 이를 통해 얻을 수 있는 이득은 다음과 같이 계산 가능하다. 페이지 크기가 2KB인 OneNAND 플래시에서 페이지의 적재에 필요한 시간 T_{load}가 페이지 읽기에 필요한 시간 T_{read}보다 크거나 같다고 가정했을 때, 이론적으로 2×N 개의 OneNAND 페이지를 그림 5에서와 같은 4KB I/O 세그먼트 단위로 끊어 읽을 때의 시간 $T_{p_segment}$와 가상 I/O 세그먼트를 이용해 읽는 시간 $T_{v_segment}$는 각각 다음과 같다.

$$T_{p\_segment}(2 \times N) = (2 \times T_{load} + T_{read}) \times N$$

$$T_{v\_segment}(2 \times N) = 2 \times T_{load} \times N + T_{read}$$

이에 따라 가상 I/O 세그먼트를 통해 얻을 수 있는 이득 $T_{benefit}$은 다음과 같다.

$$T_{benefit}(2 \times N) = T_{p\_segment}(2 \times N) - T_{v\_segment}(2 \times N) = (N - 1) \times T_{read}$$

이러한 결과는 가상 I/O 세그먼트를 이용해서 성능상의 이득을 얻기 위해서는 최소한 3개 이상의 OneNAND 페이지를 읽어야 하며 이때 얻을 수 있는 이득은 읽기 요청의 크기에 비례해서 늘어남을 보여준다.

3.2 가상 I/O 세그먼트의 비용 최적화

가상 I/O 세그먼트를 사용하면 OneNAND 플래시의 이중 버퍼링을 활용한 읽기 성능 향상을 얻을 수 있다. 하지만 가상 I/O 세그먼트를 사용하기 위해서는 매 읽기 전후에 가상 I/O 세그먼트를 생성하고 파괴하는 비용을 부담해야 한다. 가상 I/O 세그먼트의 생성과 파괴에는 가상 주소 영역의 할당 및 반환, 페이지 테이블 조작, 하드웨어 캐시 및 TLB 캐시의 조작이 필요하며, 본 논문의 실험 환경에서 이러한 작업들에 필요한 비용은 약 40us~50us 정도로써 결코 무시할 수 없는 수준이었

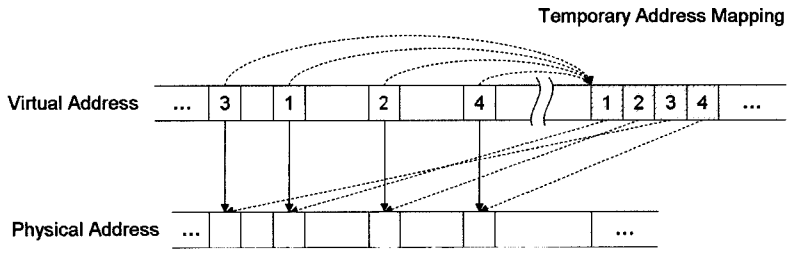

그림 7 가상 주소의 임시 맵핑을 이용한 가상 I/O 세그먼트의 생성

다. 때문에 본 연구에서는 이 비용을 줄이기 위한 최적화 기법들을 함께 연구하였다. 이 절에서는 이를 위해 제안된 3가지 최적화 기법에 대해서 설명한다.

3.2.1 주소 영역 예약 기법

주소 영역 예약(address range reservation) 기법은 특정 가상 주소 영역을 가상 I/O 세그먼트를 위해 전용하는 기법이다. 이 기법을 사용하면 가상 I/O 세그먼트 생성 및 파괴시 일어나는 가상 주소 공간의 할당 및 반환에 필요한 비용을 제거할 수 있다.

가상 I/O 세그먼트의 생성은 다음과 같은 과정을 거친다.

- 가상 메모리 계층으로부터 가상 I/O 세그먼트를 위한 가상 주소 영역을 할당
- 페이지 테이블을 조작하여 요청된 I/O 세그먼트들을 할당 받은 가상 주소 영역에 맵핑
- 가상 주소 캐시의 앨리어싱(virtual cache aliasing) 문제를 방지하기 위해서 하드웨어 캐시의 해당 주소 공간에 해당하는 영역을 플러시(flush)

가상 I/O 세그먼트의 파괴는 역으로 다음과 같은 과정을 거친다.

- 하드웨어 캐시의 내용을 메인 메모리로 플러시
- 페이지 테이블을 조작하여 가상 I/O 세그먼트와 I/O 세그먼트들의 연결을 삭제
- 해당 주소 영역의 TLB 캐시를 플러시
- 사용된 가상 주소 영역을 가상 메모리 계층에 반환

가상 주소 영역 예약 기법을 사용하면 이 모든 과정 중 첫번째와 마지막 단계, 즉 가상 주소 영역의 할당과 반환 과정을 매번 반복할 필요가 없으므로 가상 I/O 세그먼트 사용시의 비용을 줄일 수 있다. 본 논문의 실험 환경에서 이 기법을 통해서 평균 10us의 성능 향상을 얻는 것이 가능하다.

3.2.2 가상 I/O 세그먼트 생성 비용의 은닉

두번째 최적화 기법은 OneNAND 플래시의 적재 시간을 활용해 가상 세그먼트 생성 비용을 은닉하기 위한 기법이다. 그림 8에서와 같이 일반적인 가상 I/O 세그먼트의 사용에서는 가상 I/O 세그먼트 생성, 저수준 디바이스 드라이버를 통한 읽기, 가상 I/O 세그먼트 파괴의 세 단계가 순차적으로 수행된다. 이 방식의 문제점은 가상 I/O 세그먼트의 생성이 끝나고 OneNAND 플래시의 첫번째 페이지가 적재될 때까지 CPU가 대기해야 한다는 것이다. 이 때 첫번째 OneNAND 페이지의 적재와 가상 I/O 세그먼트의 생성을 병렬적으로 수행한다면 보다 효율적인 처리가 가능할 것이다. 이와 같은 개념이 그림 9에 표현되어 있다. 가상 I/O 세그먼트를 생성하기 전에 먼저 첫번째 OneNAND 페이지의 적재를 요청하고, 적재 작업이 이루어지는 동안 가상 I/O 세그먼트의

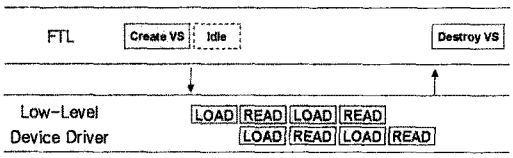

그림 8 일반적인 가상 I/O 세그먼트의 사용

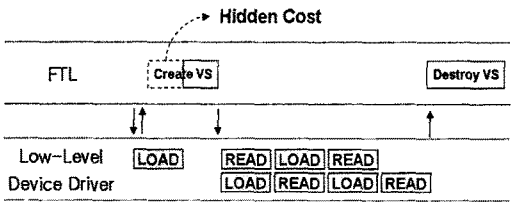

그림 9 가상 I/O 세그먼트 생성 비용의 은닉

생성 작업을 수행한다. 이 경우 OneNAND 적재 작업과 가상 I/O 세그먼트의 생성이 동시에 이루어지므로 실질적으로 가상 I/O 세그먼트 생성에 필요한 비용을 절감하는 효과를 얻을 수 있다.

3.2.3 읽기 요청의 크기에 따른 가상 I/O 세그먼트의 선택적 사용

마지막으로 언급할 최적화 기법은 읽기 요청의 크기에 따라 가상 I/O 세그먼트의 사용 여부를 선택하는 것이다. 3.1절에서 언급했듯이 가상 I/O 세그먼트로 인한 성능 이득은 읽기 요청의 크기에 비례해서 증가한다. 가상 I/O 세그먼트의 생성 및 파괴 비용 역시 요청의 크기에 비례하지만 성능 이득의 변화와 비교해서 그 초기값이 크고 증가율이 작은 특징을 보인다. 실험 결과에서 설명하겠지만, 본 논문의 실험 환경에서 읽기 요청의 크기가 8KB에서 32KB까지 변할 때 가상 I/O 세그먼트 생성 및 파괴 비용은 40us에서 50us로 증가했다. 여기에 앞서 설명한 두 최적화 기법들을 사용할 경우 이 비용은 20us 정도 절감되어 20~30us로 줄어든다. 하지만 동일한 실험환경에서 4KB의 데이타를 읽는데 필요한 시간이 90us 정도에 불과하다는 것을 고려하면 이와 같은 비용은 여전히 큰 것이다. 또한 가상 주소 조작시의 하드웨어 캐시 플러시로 인한 측정할 수 없는 성능 저하까지 고려하면 읽기 요청의 크기가 작은 경우에는 가상 I/O 세그먼트의 사용 비용이 이득을 상쇄하는 경우가 발생하게 된다.

때문에 가상 I/O 세그먼트의 사용은 그를 통해 얻을 수 있는 이득이 비용이 초과하는 경우로만 한정되어야 한다. 이는 읽기 요청의 크기가 특정 기준값보다 큰 경우에만 가상 I/O 세그먼트를 이용하도록 함으로써 간단히 구현할 수 있다. 이를 가상 I/O 세그먼트의 선택적 사용 기법이라고 한다.

4. 성능 평가

4.1 실험 환경

본 연구에서는 제안된 기법의 성능을 검증하기 위해 다음의 4 가지 플래시 기반 블록 디바이스 드라이버를 구현하고 세부적인 성능 평가를 실시하였다.

- **Original** : MTD 서브시스템에서 기본적으로 제공하는 블록 디바이스. 저수준 드라이버 호출이 섹터 단위(512bytes)로 이루어지므로 실질적으로 이중 버퍼링이 사용되지 않는다.
- **DualOP-4KB** : Original에 I/O 세그먼트 단위 이중 버퍼링 기법을 추가로 구현한 디바이스. 저수준 드라이버에 전달되는 I/O요청의 크기를 I/O 세그먼트 단위로 수정하여 최고 4KB 단위의 이중 버퍼링을 수행할 수 있도록 하였다.
- **V-Segment** : 가상 I/O 세그먼트를 이용하도록 수정된 블록 디바이스. DualOP-4KB에 가상 I/O 세그먼트를 생성/파괴하는 과정을 구현하였고, 가상 I/O 세그먼트 단위로 이중 버퍼링이 수행되도록 하였다.
- **Optimized** : V-Segment에 3.2절에서 설명한 3가지 최적화 기법을 추가로 적용한 디바이스 드라이버. 가상 I/O 세그먼트를 사용하기 위한 기준값은 12KB로 설정하였다.

실험에 사용된 각 디바이스 드라이버들은 Linux 커널 2.6.21 버전의 MTD 서브시스템과 OneNAND 저수준 드라이버를 기반으로 구현되었으며, 성능 측정은 TI OMAP 2420[16] 기반의 실험용 보드에서 이루어졌다. 저장장치로는 SAMSUNG OneNAND KF1G16Q2M-DEB6[1] 플래시 메모리를 사용하였다. 이 플래시 메모리의 삭제 블록과 페이지 크기는 각각 128KB와 2KB이며 커널의 저수준 드라이버에서 측정한 평균 성능은 표 2와 같았다.

4.2 읽기 성능 분석

첫번째 실험에서는 각 디바이스 드라이버들이 다양한 크기의 읽기 요청을 처리하는데 걸리는 시간을 측정하였다. 읽기 요청의 크기가 4KB에서 64KB까지 변하는 동안 각 디바이스 드라이버에서의 읽기 성능 변화를 측정하였으며, 그 결과가 그림 10에 표시되어 있다. 모든 영역에서 Original이 가장 나쁜 성능을 보였으며 Optimized가 가장 좋은 성능을 보였다. 요청의 크기가 64KB일 때 Optimized의 읽기 지연시간은 Original의 46%, DualOP-4KB의 72%에 불과했다. V-Segment는 읽기 요청의 크기가 16KB이상일 때는 Optimized와 비슷한 성능을 보였지만 읽기 요청의 크기가 12KB 이하일 때는 DualOP-4KB보다도 나쁜 성능을 보였다. 이는 12KB 이하의 읽기 요청 처리시에는 가상 I/O 세그먼트의 사용이 비효율적임을 말해준다.

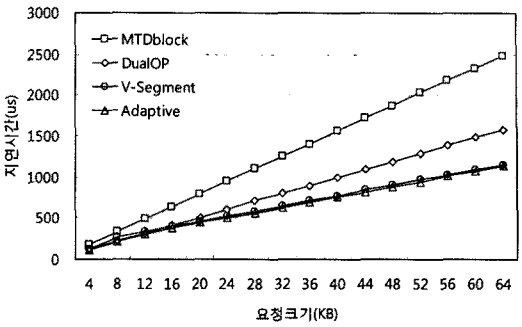

그림 10 읽기 지연시간 비교

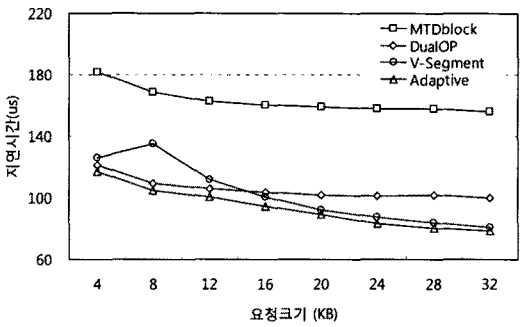

그림 11 읽기 처리시 4KB 단위 비용 비교

그림 11은 그림 10의 결과로부터 4KB의 읽기를 처리하는데 필요한 단위 비용을 계산한 결과이다. 그림은 각 디바이스 드라이버의 읽기 성능 차이를 보다 명확하게 보여주므로, 요청의 크기가 4KB~12KB일 때와 16KB 이상일 때 V-Segment와 DualOP-4KB의 성능 역전 현상을 확인할 수 있다. V-Segment의 4KB 단위 비용은 요청 크기가 4KB~12KB일 때는 비교적 크지만, 요청 크기가 증가함에 따라 점점 감소하여 요청의 크기가 16KB 이상일 때는 Optimized에 근접할 만큼 작아진다. Optimized는 3.2절에서 언급한 최적화 기법의 효과로 인해 어떤 경우에도 다른 디바이스 드라이버들에 비하여 좋은 성능을 보인다.

4.3 비용 분석

두번째 실험에서는 가상 I/O 세그먼트의 사용에 필요한 비용의 변화를 측정하였다. 읽기 요청 처리에 수반되

표 2 KF1G16Q2M-DEB6 플래시 메모리의 성능 측정 결과

작업	읽기(Sync. Mode)	쓰기(Async. Mode)	페이지 적재	페이지 프로그램
시간	28us	80us	32us	199us

는 비용은 크게 두 가지로 나눌 수 있다. 첫 번째는 범용 블록 계층과 관련된 비용이고 두 번째는 가상 I/O 세그먼트의 생성과 삭제에 관련된 비용이다. 이 중에서 범용 블록 계층과 관련된 비용은 요청의 준비(get request)에 필요한 비용과 처리된 I/O 요청을 종료(end request)하는데 필요한 비용으로 나뉜다. 요청의 준비는 I/O 요청 큐로부터 I/O 요청을 가져오는 과정을 말하며 I/O 요청의 종료는 해당 I/O 요청에 블록되어 있는 프로세스를 깨우고(wake-up) I/O 요청 구조체를 파괴하는 과정을 말한다.

그림 12는 1KB 버퍼를 이용하여 8KB부터 32KB까지의 읽기 요청을 처리할 때 각 디바이스 드라이버들의 비용 변화를 요약한 그래프이다. 그래프의 Original, DualOP-4KB는 해당 디바이스 드라이버의 범용 블록 계층에 관련된 비용, 즉 요청의 준비와 종료에 소모되는 비용의 변화를 보여준다. 가상 I/O 세그먼트 기법의 비용은 3가지로 나누어 표시되어 있다 첫 번째 V-Seg-Block은 V-Segment가 요청의 준비와 종료에 소모하는 비용을 나타내고, V-Seg-Vmap은 가상 I/O 세그먼트의 생성과 삭제에 필요한 비용을 나타낸다. 마지막으로 V-Seg-Total은 이 둘을 합한 전체 비용을 뜻한다.

먼저 요청의 준비와 종료에 필요한 비용을 살펴보면 Original과 DualOP-4KB의 경우 요청 크기 증가에 따라 비용이 급격하게 증가함을 볼 수 있다. 반면 가상 I/O 세그먼트 기법에서의 요청 준비와 종료 비용인 V-Seg-Block은 그 초기값이 다른 두 기법에 비해 조금 큰 반면 요청 크기의 증가에 따른 변화는 매우 작다. 때문에 요청의 크기가 8KB일 때를 제외하면 두 기법보다 낮은 비용을 보여준다. 이러한 차이가 나타나는 것은 각 기법에서 I/O요청의 준비와 완료를 처리하는 방식 때문이다. 가상 I/O 세그먼트를 사용하는 경우에는 I/O 요청 준비 및 종료 작업을 가상 I/O 세그먼트를 이용한 읽기 요청 처리 전후에 한번에 처리한다. 반면 Original과 DualOP-4KB는 동일한 작업을 각각의 I/O 세그먼트 처리시마다 조금씩 나누어 처리한다. 때문에 동일 작업의 반복으로 요청 준비, 종료 비용이 급격하게 증가하는 것이다. 다음으로 가상 I/O 세그먼트의 생성과 파괴에 필요한 비용인 V-Seg-Vmap은 요청의 크기 증가에서 따라 40us에서 최대 50us까지 완만하게 증가한다.

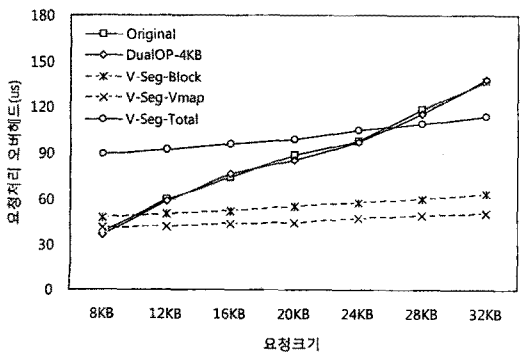

그림 12 비용 비교

가상 I/O 세그먼트 사용시의 실제 비용인 V-Seg-Total은 이들 두 가지 비용의 합이다. 이 값은 8KB부터 24KB까지 다른 두 기법보다 큰 수치를 보이지만 요청의 크기가 커질수록 그 차이가 줄어들어서 28KB 이상일 때는 오히려 다른 디바이스 드라이버들 보다 작아진다. 이와 같은 사실은 가상 I/O 세그먼트 기법이 작은 읽기 요청의 처리시 비효율적인 이유를 설명해준다.

4.4 최적화 기법의 효과 분석

그림 13은 Optimized와 V-Segment의 성능차이를 나타낸 표로서 3.2절에 소개된 최적화 기법들의 효과를 보여준다. Optimized는 가상 I/O 세그먼트 기법을 선택적으로 사용하므로 효율이 떨어지는 4KB~12KB 구간에서는 가상 I/O 세그먼트를 사용하는 대신 4KB 크기의 I/O 세그먼트 단위로 I/O를 수행한다. 그래프의 해당 구간에서 최고 61us에 달하는 성능 차이는 가상 I/O 세그먼트를 사용하지 않는 Optimized와 가상 I/O 세그먼트를 사용하는 V-Segment의 성능 차이를 보여주는 것이다. Optimized가 가상 I/O 세그먼트를 사용하는 16KB 이상의 구간에서는 주소 영역 예약 기법(3.2.1절), 생성 비용 은닉 기법(3.2.2절)의 사용이 성능 차이를 유발하며, 이 구간에서 이들 두 최적화 기법을 통해서 얻은 성능 향상은 평균 22us였다. 이상의 결과는 본 논문에서 제안한 최적화 기법이 효과적임을 보여주는 것이다.

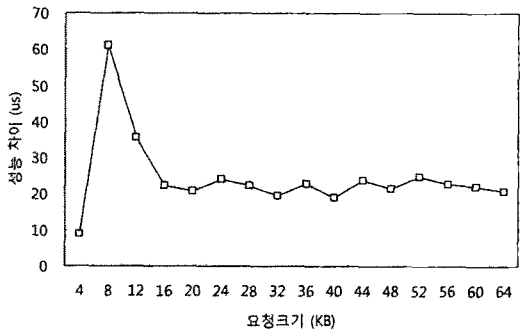

그림 13 최적화 기법의 효과

4.5 페이지 폴트 처리 성능 분석

마지막 실험에서 실제 요구 페이징 시스템에서 응용 프로그램 수행시의 페이지 폴트 처리에 걸리는 시간을 측정하였다. 이와 같은 실험을 선택한 이유는 페이지 폴

트 처리시 발생하는 읽기 요청 패턴이 모든 요구 페이징 시스템에 공통으로 존재하는 워크로드이며, 페이지 폴트 처리 성능 자체가 디바이스의 읽기 성능에 크게 좌우되기 때문이다. 또한 요구 페이징 시스템에서의 활용이 OneNAND 플래시의 대표적인 활용 분야이기 때문이다. 때문에 이 실험 결과는 실제 시스템에서 가상 I/O 세그먼트 기법의 사용을 통해 얻을 수 있는 이득을 명확히 보여줄 것이다.

페이지 폴트는 크게 메이저 페이지 폴트(major page fault)와 마이너 페이지 폴트(minor page fault)의 두 종류로 나뉜다. 메이저 페이지 폴트는 디바이스의 읽기를 수반하는 페이지 폴트를 말한다. 파일 혹은 스왑 영역으로부터 폴트를 유발한 페이지를 읽는 경우가 이에 해당하며, 그 처리 성능은 디바이스의 읽기 성능에 크게 좌우된다. 마이너 페이지 폴트는 메이저 폴트를 제외한 나머지 형태의 페이지 폴트로써, 디바이스의 읽기를 수반하지 않으며 보통 매우 짧은 시간 안에 처리된다. 일반적으로 메이저 페이지 폴트는 빈도 면에서 전체 페이지 폴트 발생의 극히 일부분을 차지하지만 처리 시간 면에서는 전체 페이지 폴트 처리 시간의 대부분을 차지한다.

실험에서는 내장형 시스템의 대표적인 GUI 환경인 Qtopia[17]를 부팅시킨 후 부팅이 완료될 때까지 발생하는 페이지 폴트들의 처리 시간을 측정하였다. 실험은 각각의 디바이스 드라이버 위에 EXT3 파일 시스템을 읽기 전용으로 마운트한 상태에서 수행되었으며, 디바이스 드라이버의 종류에 관계없이 총 110번의 메이저 페이지 폴트와 5,643번의 마이너 페이지 폴트가 발생했고 총 1,929개의 페이지(7.5MB)에 대한 읽기 요청이 처리되었다.

그림 14는 측정된 결과를 전체 페이지 폴트 처리 시간과 메이저 페이지 폴트 처리 시간으로 분류하여 보여준다. 디바이스 읽기 성능의 영향을 직접적으로 받는 메이저 페이지 폴트 처리 성능에서는 Optimized가 178ms로 가장 좋은 성능을 보였고 V-Segment가 4ms 차이로 그 뒤를 따랐다. Optimized의 메이저 페이지 폴트

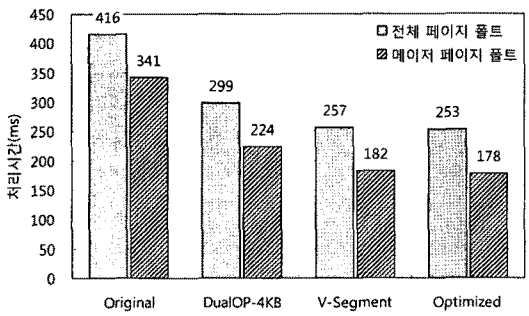

그림 14 페이지 폴트 처리 시간 비교

처리 시간은 Original의 52%에 불과하며 DualOP-4KB에 비해서도 79% 수준임을 알 수 있다. 전체 페이지 폴트 처리 성능의 변화 역시 메이저 폴트 처리 성능의 변화와 동일하며, Optimized가 Original, DualOP-4KB에 비해서 각각 61%, 85%의 처리 시간을 보였다. 모든 경우에서 전체 페이지 폴트 처리 시간과 메이저 페이지 폴트 처리 시간의 차이는 동일하며, 이는 가상 I/O 세그먼트의 사용이 마이너 페이지 폴트 처리 성능에는 어떤 영향도 미치지 않는다는 사실을 보여준다. 이상의 실험 결과로부터 본 논문에서 제시한 가상 I/O 세그먼트 기법이 OneNAND 플래시의 읽기 성능 및 OneNAND 플래시 기반 요구 페이징 시스템의 성능 향상에 효과적임을 알 수 있다.

5. 결 론

본 연구에서는 현재의 블록 I/O 계층 구조 내에서 OneNAND 플래시 메모리의 높은 읽기 성능을 활용할 수 있는 기법인 가상 I/O 세그먼트의 개념과 최적화 기법을 제안하였고, 실제 구현을 통한 실험으로 그 성능과 유용함을 증명하였다.

실험 결과 본 논문에서 제안한 가상 I/O 세그먼트와 관련 최적화 기법은 OneNAND 플래시의 읽기 지연시간을 최고 46%까지 감소시켰으며, 실제 요구 페이징 시스템에서의 실험에서도 메이저 페이지 폴트 처리시간을 52%까지 감소시키는 좋은 결과를 보였다. 이러한 결과는 본 논문에서 제안한 기법이 향후 OneNAND 플래시 메모리 기반의 고성능, 저비용의 휴대용 정보기기의 개발에 효과적으로 이용될 수 있음을 보여준다.

참 고 문 헌

[1] OneNANDTM Features and Performance, 2005 http://www.samsung.com/global/business/semiconductor/products/fusionmemory/Products_ProductOverview.html

[2] M. Santarini, "NAND versus NOR: Which flash is best for bootin' your next system?," in Electronics, Design, Strategy, News (EDN). vol. 21, 2005, pp. 41-48.

[3] J. In, I. Shin, and H. Kim, "SWL: a search-while-load demand paging scheme with NAND flash memory," in LCTES'07, San Diego, California, USA, 2007, pp. 217-226.

[4] E. Gal and S. Toledo, "Alogorithms and Data Structures for Flash Memories," ACM Computing Surveys, vol. 37, pp. 138-163, 2005.

[5] 이수관, 민상렬, 조유근, "플래시 메모리 관련 최근 기술 동향," 정보과학회지, vol. 24, pp. 99-106, 2006.

[6] 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 ISLPED, California, USA, 2004, pp. 338-343.
[7] H. Jung, K. Yoon, H. Shim, S. Park, S. Kang, and J. Cha, "LIRS-WSR: Integration of LIRS and Writes Sequence Reordering for Flash Memory," Lecture Notes in Computer Science, vol. 4705, pp. 224-237, 2007.
[8] Y. Yoo, H. Lee, Y. Ryu, and H. Bahn, "Page Replacement Algorithms for NAND Flash Memory Storages," Lecture Notes in Computer Science, vol. 4705, pp. 201-212, 2007.
[9] D. Jung, J.-S. Kim, S.-Y. Park, J.-U. Kang, and J. Lee, "FASS: A Flash-Aware Swap System," in International Workshop on Software Support for Portable Storage (IWSSPS), 2005.
[10] S. Park, H. Lim, H. Chang, and W. Sung, "Compressed Swapping for NAND Flash Memory Based Embedded Systems " LNCS, vol. 3553, pp. 314-323, 2005.
[11] Y. Joo, Y. Choi, C. Park, S. W. Chung, E. Y. Chung, and N. Chang, "Demand Paging for OneNANDTM Flash eXecute-in-place," in Proceedings of CODES+ISSS'06, Seoul, Korea, 2006, pp. 229-234.
[12] Y. Joo, J. Park, S. W. Chung, E. Y. Chung, and N. Chang, "Delayed Dual Buffering: Reducing Page Fault Latency in Demand Paging for OneNAND Flash Memory," 전자공학회 논문지, vol. 44, pp. 270-278, 2007.
[13] Samsung OneNANDTM KFXG16Q2M-DEB6 data sheets, http://www.samsung.com/global/business/semiconductor/productInfo.do?fmly_id=160&partnum=KFH1G16Q2M
[14] Memory Technology Device Subsystem for Linux - homepage, http://www.linux-mtd.infradead.org
[15] Daniel P. Bovet, and Marco Cesati, Understanding the Linux Kernel, 3rd Ed., p.560, O'reilly, 2006
[16] TI OMAP 2420 Platform, http://focus.ti.com/general/docs/wtbu/wtbuproductcontent.tsp?templateId=6123&navigationId=11990&contentId=4671
[17] Qtopia GUI Platform, http://trolltech.com/products/Qtopia.

현 승 환

2000년 및 2002년에 서울대학교 컴퓨터 공학과에서 학사 및 석사 학위를 받았다. 현재 서울대학교 컴퓨터 공학과에서 박사과정을 밟고 있으며, 플래시 메모리 및 내장형 시스템, I/O 서브시스템에 관심을 가지고 있다.

고 건

1974년 서울대학교에서 응용 물리학 학사 학위를 받았으며, 1979년과 1981년에 미국 버지니아 대학에서 석사 및 박사 학위를 받았다. 1981년부터 1983년까지 AT&T사의 Bell 연구소에서 연구원으로 근무했으며, 1983년부터 현재까지 서울대학교 전기.컴퓨터공학부 교수로 재직하고 있다. 관심 연구 분야는 운영체제 및 내장형 시스템, 컴퓨터 시스템 성능 분석이다.