Journal of Digital Contents Society Vol. 11 No. 1 Mar. 2010(pp. 91-97)

페이지 주소 캐시를 활용한 NAND 플래시 메모리 파일 시스템에서의 효율적 주소 변환 테이블 관리 정책

김정길*

요 약

비휘발성, 저전력 소모, 안정성 등의 장점을 가진 NAND 플래시 메모리는 고집적화, 대용량화, 저가격화를 통하여 다양한 디지털시스템의 데이터 저장장치로 사용되고 있다. 플래시 메모리의 다양한 분야에서의 응용 확대와 동시에 플래시 메모리의 대용량화는 플래시 메모리의 주소 변환 테이블의 전체 크기를 증가시켜 SRAM에 저장하기에 용량이 부족한 문제점을 발생시킨다. 본 논문에서는 하이브리드 변환 기법 기반의 플래시 메모리 파일 시스템에서 페이지 주소 캐시를 이용한 효율적인 주소 테이블 관리 정책을 제안한다. 제안하는 기법은 다양한 메타 데이터 기반의 전체 테이블의 정보를 맵 블록을 이용하여 효율적으로 통합 관리함으로써 높은 성능을 유지할 수 있다. PC 환경에서의 다양한 응용프로그램을 실험한 결과 제안하는 페이지 주소 캐시는 2.5% 이하의 낮은 미스율로 높은 효율성을 유지하며 전체 쓰기 연산 요청에서 평균 33%의 실제 쓰기 연산의 실행으로 전체 쓰기 연산에서 발생하는 오버헤드를 줄여 주었다.

An Efficient Address Mapping Table Management Scheme for NAND Flash Memory File System Exploiting Page Address Cache

Cheong Ghil Kim*

Abstract

Flash memory has been used by many digital devices for data storage, exploiting the advantages of non-volatility, low power, stability, and so on, with the help of high integrity, large capacity, and low price. As the fast growing popularity of flash memory, the density of it increases so significantly that its entire address mapping table becomes too big to be stored in SRAM. This paper proposes the associated page address cache with an efficient table management scheme for hybrid flash translation layer mapping. For this purpose, all tables are integrated into a map block containing entire physical page tables. Simulation results show that the proposed scheme can save the extra memory areas and decrease the searching time with less 2.5% of miss ratio on PC workload and can decrease the write overhead by performing write operation 33% out of total writes requested.

Keywords : NAND flash memory, file translation layer, hybrid address mapping, cache

1. 서론

반도체 설계 및 공정 기술의 발달은 휴대용 통신 및 멀티미디어 기기의 고사양화와 더불어 대용량의 저장장치를 필요로 하게 하였다. 최근의 스마트폰(smartphone) 및 태블릿(tablet) PC의 대중화는 이러한 변화를 선도하고 있다. 그 결과 이러한 기기들의 저장장치는 기존의 디스크 기반 저장장치를 대체하여 전원이 꺼지더라도 저장 데이터가 유지 가능한 비휘발성의 플래시 메모리를 사용하고 있다. 이러한 플래시 메모리는 하드디스크(HDD)에 비해 크기가 작고 가

※ 제일저자(First Author) : 김정길
접수일:2010년 02월 16일, 완료일:2010년 03월 30일
* 남서울대학교 컴퓨터학과
cgkim@nsu.ac.kr
▣ 본 연구는 2008학년도 남서울대학교 연구처 교내 연구비 지원에 의해 수행되었음

벼우며, 전력을 적게 사용하고, 데이터 접근 성능이 뛰어나며, 충격에 강한 장점을 가지고 있다. 그러나 플래시 메모리는 기존의 저장장치인 하드디스크와 물리적인 다른 점 외에 동작적인 차이점을 가지고 있다.

플래시 메모리는 읽기(read), 쓰기(write) 연산 이외에 소거(erase) 연산을 필요로 한다. 기본적으로 읽기/쓰기 연산은 페이지 단위로 이루어지나, 기존의 데이터를 갱신하여 다시 쓰기 연산을 하기 위해서는 해당 페이지의 소거 연산을 먼저 수행해야 하며, 이때 소거 연산의 단위는 페이지보다 큰 단위인 블록이기 때문에 특정 페이지의 데이터를 갱신하기 위해서는 동일 블록에 속한 다른 페이지들도 소거되어야 한다. 즉, 데이터의 직접 갱신(in-place update)이 불가능하다. 각 연산은 처리 속도에도 큰 차이가 있어 일반적인 NAND 플래시 메모리에서 페이지의 읽기와 쓰기는 각각 25us, 200us인 반면에 블록 소거의 시간은 2ms가 소요된다. 또한 플래시 메모리의 각 블록은 제한된 횟수의 소거와 쓰기만을 허용한다.

플래시 메모리의 이러한 특징들로 인하여 플래시 메모리를 사용하는 시스템에서는 사용자에게 기존의 저장장치들과 동일한 장치로 인식되게 하기 위해 별도의 컨트롤러와 플래시 변환 계층(flash translation layer, FTL)이라고 불리는 시스템 소프트웨어를 필요로 한다[1]. FTL은 파일 시스템과 플래시 메모리 사이에 위치하여 파일 시스템에서 사용하는 상위 계층의 논리 주소(logical address)를 실제 플래시 메모리의 블록 번호 또는 페이지 번호와 같은 물리 주소(physical address)로 변환하는 주소변환(address translation 또는 address mapping)과 덮어쓰기(overwrite) 연산이 발생하여도 사용자가 소거 연산을 알 수 없게끔 감추어주는 역할을 한다.

주소 변환 방법에는 페이지 변환(page mapping), 블록 변환(block mapping), 그리고 이 두 방법을 혼용해서 사용하는 하이브리드 변환(hybrid mapping) 방법이 있다. 그러나 플래시 메모리의 대용량화는 해당 변환 테이블의 크기를 증가시키며, 독립적인 사상 블록을 유지할 경우 추가적인 접근 시간이 소요되는 단점이 있다. 따라서 본 논문에서는 하이브리드 변환 방법 기반의 FTL의 성능 향상을 위한 방법을 제안하고 있다. 이를 위하여 페이지 주소 캐시(page address cache)를 이용하여 효율적으로 여러 테이블을 운용하며, 해당 정보는 일반 플래시 메모리들과 동일하게 플래시 메모리에 저장된다.

본 논문의 구성은 다음과 같다. 2장에서는 이제까지 제시된 플래시 메모리의 주소 변환 기법을 간략히 소개한다. 3장은 제안하는 페이지 주소 캐시를 이용한 효율적 테이블 관리 방법을 기술한다. 4장에서는 시뮬레이션을 사용하여 제시한 방법의 성능을 평가하고, 마지막으로 5장에서 결론을 보인다.

2. 관련연구

앞 장에서 언급했듯이 플래시 메모리는 데이터의 직접 갱신이 불가능한 제약으로 인하여 특정 주소에 대해 데이터 갱신이 요청되는 경우 기존 데이터가 기록되어 있던 물리 주소가 아닌 사용 되지 않고 있는 다른 물리 주소에 갱신하고자 하는 데이터가 기록될 수 있다. 이 때 논리 주소에 대한 물리 주소가 변경되더라도 동일한 논리 주소로 계속 접근이 가능하도록 사상 정보를 유지해야 한다. 일반적으로 이를 위하여 사상 정보를 동일한 페이지 내의 데이터 영역과 스페어 영역에 위치하는 방법[2]과 별도의 테이블 구조를 이용하는 방법[3] 2가지가 있다. 이때 기록 단위가 페이지 단위 또는 블록 단위 여부에 따라 각각 페이지 수준 변환(page mapping)[4]과 블록 수준 변환(block mapping)[3][5]으로 구분되며, 이 두 방법을 혼용해서 사용하는 하이브리드 변환(hybrid mapping)[6] 방법이 있다.

블록 변환은 주소 변환 테이블을 블록 단위로 관리하며, 블록 내부에서의 페이지 위치는 논리 주소와 물리 주소가 동일하게 기록된다. 그러므로 페이지를 찾을 때 블록 단위로 주소를 변환하여 페이지에 접근하며 변환 테이블을 유지하기 위하여 저장 공간이 많이 요구되지 않는다. 그러나 한 페이지에 대한 갱신 요청에도 새로운 블록을 할당받아 블록 내의 다른 페이지들을 모두 복사해야하므로 쓰기 연산의 성능이 저하 된다. 페이지 변환 방식에서는 페이지 단위로 논리 주소와 물리 주소 사이의 주소 변환 정보를 관리하며, 그 결과 블록의 임의의 위치에 페이지를

저장할 수 있으며. 데이터를 갱신하는 경우 이전 데이터를 무효화(invalidate)시키고 새로운 페이지만을 기록하기 때문에 연산 속도가 블록 변환 방식 보다 빠르다. 그러나 모든 페이지의 변환 주소를 테이블로 관리하기 때문에 많은 저장 공간이 필요로 하는 문제점을 가지고 있다. 하이브리드 변환 방식은 페이지 변환과 블록 변환의 두 기법을 동시에 사용함으로 두 방식의 단점을 보완 한다. 이 경우 블록 변환 테이블은 물리적인 주소를 결정하고 특정 페이지는 결정된 블록내의 페이지 테이블을 이용하여 결정 된다. 또 다른 기법의 하이브리드 변환 방법으로 블록단위 연관기법(block associative sector translation, BAST)[7]이 제안되었다. BAST는 플래시 메모리의 블록을 데이터를 저장하기 위한 데이터 블록과 덮어쓰기 연산이 발생할 때 사용하는 로그블록으로 구분하였다. BAST는 데이터 블록에 대해서는 블록단위 사상을 하고, 로그블록에 대해서는 섹터단위 사상을 한다. 한편 BAST의 로그블록 활용률이 떨어지는 단점을 보완하기 위해 모든 데이터블록이 로그블록을 공유하는 완전 연관기법(fully associative sector translation, FAST)[8]도 제안되었다. FAST는 기존 BAST에서의 로그블록을 순차쓰기 로그블록과 임의쓰기 로그블록으로 나누어 사용한다.

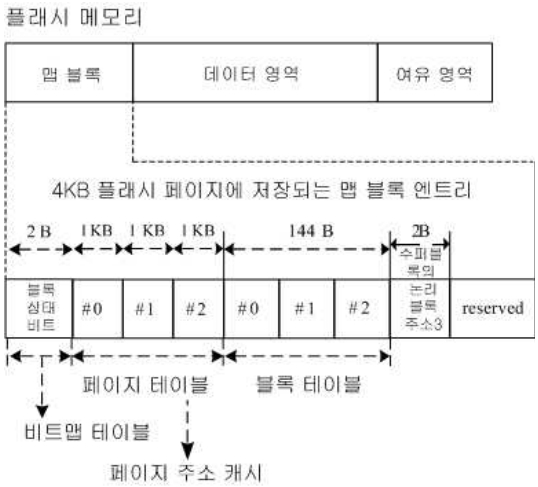
3. 페이지 주소 캐시 기반의 주소 변환 테이블 관리 정책

본 논문에서는 블록 변환 테이블을 이용하여 물리적 주소를 결정하고 특정 페이지를 이용하여 결정된 블록내의 페이지 테이블을 이용하는 하이브리드 주소 변환(hybrid mapping) 기법 기반의 FTL의 성능 향상을 위하여 페이지 주소 캐시를 이용한 효율적 테이블 관리 기법을 제안하고 있다. 이를 위하여 필요한 모든 테이블을 하나의 통합된 맵 블록(map block)에 저장하고, 맵 블록은 제안된 캐시를 이용하여 효율적으로 접근 가능하며, 초기 스캔 작업을 통하여 필요한 테이블들은 초기화 된다.

플래시 메모리는 블록단위로 나뉘어져 있는데, 대 블록 단위인 경우 각 블록은 128개의 페이지로 구성되며 각 페이지는 4KB의 데이터 영역과 128B의 여유 공간으로 구성이 되어 있다. 플래시 메모리에 대한 참조 명령은 논리적 주소와 필요한 데이터의 길이를 포함하고 있는데, 참조된 주소는 FTL에 의해 주소 변환 과정을 거친다. 제안하는 시스템에서 변환 작업을 위하여 여러 종류의 메타 테이블이 사용된다. 본 논문에서 제안하는 FTL에서는 여러 종류의 테이블을 통합하여 맵 블록을 구성한 후 페이지 주소 캐시를 이용하여 전체 테이블을 효율적으로 관리하게 된다. 필요한 여러 종류의 테이블을 플래시 메모리의 주요 연산 수행 과정을 통해 살펴보면 다음과 같다

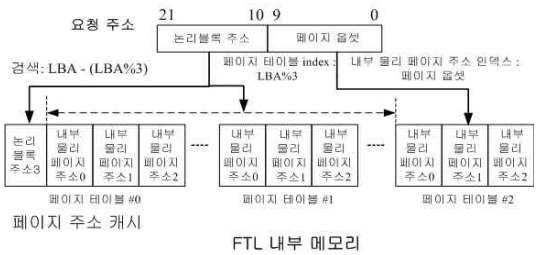
읽기(read)는 플래시 메모리의 특정 페이지를 참조하는 연산으로 이때 FTL은 블록 테이블을 참조하여 물리 블록 주소를 얻고, 페이지 테이블을 참조하여 해당 블록의 페이지 옵셋 값을 얻는다. 쓰기(write) 연산은 내부 페이지에서 순차적으로 진행되어야 하기 때문에 각 블록에서 최근에 갱신된 페이지 포인터(pointer of recently updated page, P-RUP) 정보를 저장하고 있어야 한다. 쓰기 연산이 요청되었을 경우 포인터 값은 증가되며 페이지 테이블 내의 해당 물리 주소는 갱신된 주소로 변경되어야 한다. 또한 쓰기 연산은 새로운 미사용 블록의 요구를 동반하는 합병(merge) 연산을 필요로 한다. 따라서 FTL은 각 물리 주소의 삭제 여부에 대한 정보를 유지하여야 한다. 해당 블록을 선택한 후에는 합병 연산을 수행하기 위하여 정해진 순서의 읽기와 쓰기 연산을 수행한다.

제안된 시스템에서는 이러한 모든 테이블들을 통합하여 플래시 메모리의 특정장소에 저장하였으며 이를 맵 블록이라고 부른다. (그림 1)은 8개의 뱅크로 구성된 10GB의 플래시 메모리 시스템에서의 맵 블록을 보여이고 있다. 특히 최근의 플래시 메모리는 다수의 블록을 결합하여 상위의 수퍼 블록[9] 을 구성하여 다중 뱅크 접근(multi-bank access)이 효과적으로 인터리빙 될 수 있도록 하고 있다. 본 논문에서도 하나의 수퍼 블록을 8개 뱅크 기반의 8개 블록으로 구성하였으며, 가상적으로 128*3개의 페이지로 구성된 단일 블록으로 처리하였다. 하나의 맵 블록 주소(map block address)는 4KB 크기의 플래시 메모리 페이지에 저장되며 각 필드의 구성과 의미는 다음과 같다.



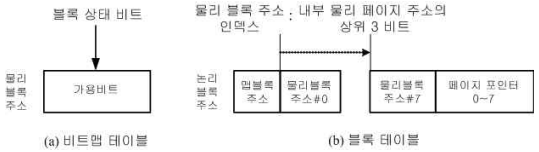
(그림 1) 플래시 메모리 및 맵 블록 구조

한 개의 맵 블록 엔트리의 물리 주소인 맵 블록 주소는 비트맵 테이블 생성을 위한 블록 상태 비트 필드, 3개의 페이지 테이블 필드, 3개의 블록 테이블 필드, 수퍼 블록의 논리 블록 주소 필드로 나누어진다. (그림 1)은 4KB 크기의 하나의 맵 블록 엔트리의 필드 구성과 각 필드의 사이즈를 보이고 있다. 여기서 블록 상태 비트 필드는 비트맵 테이블(그림 3)에 페이지 테이블 필드는 페이지 주소 캐시(그림 2)에 저장된다.



(그림 2) 페이지 주소 캐시의 구조 및 주소 사상 방법

다시 페이지 테이블과 비트맵 테이블의 각 필드 구성을 살펴보면 다음과 같다. (그림 2)에서 보듯, 하나의 블록은 128개의 페이지를 가지며 하나의 수퍼블록은 8개의 블록으로 구성되어 있기 때문에, 페이지 테이블은 내부적으로 128*8개의 내부 물리 페이지 주소를 가진다. 그리고 블록 테이블은 (그림 3)의 (b)에서 보여 주듯이 8개의 물리 블록 주소 필드와 최근에 갱신된 페이지 포인터 필드로 구성된다. 앞에서 언급되었듯이 한 개의 맵 블록 엔트리는 3개의 연속된 수퍼블록을 위하여 3개의 페이지 테이블과 비트맵 테이블로 구성되었다. 그러나 첫 번째 수퍼블록의 논리 블록 주소는 페이지 주소 캐시의 각 엔트리에 포함되며 나머지 필드들은 해당 옵셋 위치에서 찾을 수 있다. (그림 3)의 (a)는 블록의 가용 여부를 표시하는 비트들로 구성된 비트맵 테이블을 보여준다. 블록 테이블 및 페이지 테이블은 논리 주소를 물리 주소로 변경하는데 필요하며, 비트맵 테이블은 블록 합병 작업과 불량 블록 관리에 이용된다. 비트맵 테이블과 블록 테이블은 초기 실행 시 생성되며 페이지 테이블은 참조 시 캐시에 로드된다.



(그림 3) 비트맵 테이블과 블록 테이블 구조

제안된 시스템의 동작 모델은 초기 비트맵 테이블과 블록 테이블 생성 과정과 페이지 테이블 캐시 접근 시 적중(hit)과 미스(miss) 과정을 통하여 요약 설명된다.

비트맵 테이블과 블록 테이블의 생성 과정은 다음과 같다. 초기 시동 시 해당 테이블들을 패치(ftech)하기 위하여 맵 블록 영역에서 블록 상태 비트, 논리 블록 주소, 블록 테이블 필드들을 스캔한다. 이때 블록 테이블의 각 엔트리들은 한 개의 수퍼 블록을 위한 각각 8개의 물리 블록 주소와 최근에 갱신된 페이지 포인터, 그리고 그에 상응하는 맵 블록 엔트리를 가리키는 한 개의 역 포인터로 구성된다. 블록 상태 비트들 또한 사용 가능한 블록의 비트맵 테이블을 만들기 위하여 블록 주소와 함께 읽는다. 블록 상태 비트는 쓰기 가능, 쓰기 불가능, 불량 3개의 다른 상태를 표시한다. 모든 물리 블록 주소는 비트맵 테이블에서 가용 여부 비트를 이용하여 삭제 여부를 알려준다. 이 테이블을 이용하여 추가적인 비용 없이 불량 블록의 관리가 가능하다. 이때 불량 블록은 비트맵 테이블에서 -1을 할당함으

로 오류 주소로 인식시켜 추가적인 접근이나 할당이 발생하지 않도록 하였다.

제안하는 페이지 주소 캐시를 이용한 페이지 테이블의 접근은 다음과 같이 수행된다. 최초 테이블 생성 작업의 완료 후, 페이지 테이블의 각 엔트리들은 캐시 영역으로 패치(fetch)되며, 이때 제안된 캐시의 쓰기는 write-back 정책을 채택하였다. 제안된 페이지 주소 캐시는 소프트웨어 방식의 완전 연관 캐시(fully-associative cache)로서 FTL의 일정 부분의 물리적인 저장 공간을 필요로 하며, 캐시 블록 참조 시 순차적으로 접근된다.

제안된 시스템에서 캐시 접근 시 논리 주소를 물리 주소로 변환하는 사상 방법은 (그림 2)에서와 같이 요청 주소가 논리 주소 블록 필드와 페이지 오프셋 필드로 구분되어 처리되는 과정을 통하여 알 수 있다. 캐시 적중, 즉 캐시 접근 시 요청한 데이터가 캐시에 있는 경우 페이지 주소 필드의 논리 블록 주소(logical block address, LBA)를 캐시 태그로 탐색이 시작되며 이때 수퍼 블록의 논리 블록 주소는 LBA3로 구분 표시할 때 LBA3는 식 (1)로 구할 수 있다.

$$LBA3 = LBA - (LBA \bmod 3) \quad (1)$$

참조한 데이터를 페이지 주소 캐시 엔트리 내에서 발견한 뒤 내부 물리 페이지 주소(internal physical page address, IPPA) 값은 식 (2)를 이용하여 구한다.

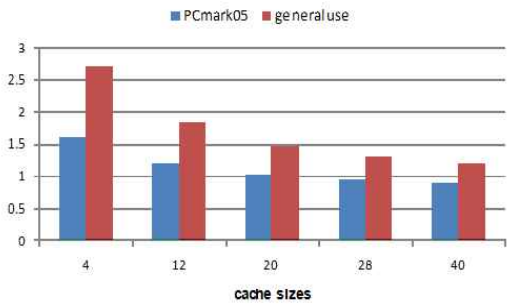
$$IPPA = (LBA \bmod 3) * PT\_size + pageoffset \quad (2)$$

결과적으로 논리 블록 주소(LBA)는 블록 테이블에서 하나의 엔트리를 선택하며 계산된 내부 물리 페이지 주소(IPPA)의 상위 3 비트를 이용하여 8개의 물리 블록 주소 가운데 한 개의 물리 블록 주소를 선택한다.

제안된 페이지 주소 캐시는 전체 맵 블록 엔트리의 전체 페이지들의 주소를 저장한다. 즉, 제안하는 페이지 주소 캐시의 각 엔트리의 크기는 3KB가 된다. 그 결과 캐시 사이즈의 증대는 캐시내의 지역적 인접성의 특성을 활용할 수 있는 기회가 증대되며 캐시를 사용함으로 얻을 수 있는 성능 향상은 크게 된다. 이때 캐시의 갱신 정책은 선입선출 방식을 사용하였다. 쓰기 정책은 해당 맵 블록의 더티 비트가 셋 되었을 경우만 플래시 메모리내의 맵 블록을 갱신한다.

4. 실험 및 결과

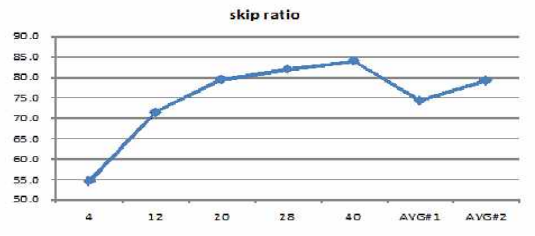
제안 페이지 주소 캐시를 이용한 FTL의 성능 검증을 위하여 본 논문에서는 캐시의 미스율(miss ratio)을 측정하였다. 동시에 제안 캐시로 인하여 발생 가능한 과부하는 전체 발생한 쓰기 연산에서 실제 write-back이 실행 된 비율 값을 구하였다. 이를 위하여 제안 FTL은 이전 연구에서 구현된 NAND 플래시 메모리 시뮬레이터[10]를 이용하여 구현하였다. 시뮬레이터는 디스크 I/O 트레이스를 이용하여 하이브리드 주소 변환 기법을 적용하였으며, 범용 PC를 위한 PCmark05[11] 벤치마크와 일반 사용자가 하루 동안 컴퓨터를 사용할 때 추출한 트레이스인 generaluse를 입력 실험 데이터로 사용하였다.



(그림 4) 캐시 사이즈 변화에 따른 미스율

(그림 4)는 2가지 실험 입력 패턴에 대하여 페이지 주소 캐시에서 발생한 미스율 값을 보여 준다. X축은 캐시 사이즈를 4KB에서부터 40KB까지 변화를 주었으며 y축은 매 1,000 입력에 의한 평균 미스율 표시한다. 여기서 제안하는 페이지 주소 캐시의 미스율은 PC 벤치마크에서 평균 1.2% 이하 그리고 generaluse에서 평균 1.8% 이하로 우수한 성능을 보인다. 해당 성능의 다른 FTL과의 비교는 각각의 연구가 자신들의 실험 데이터를 사용하여 성능을 측정하는 관계로 객관적인 비교는 어려운 실정이다. 그러나 제안하는 페이지 주소 캐시는 다른 방법들[9]의 5%대 미스율과 비교하여 상대적으로 낮은 1.29% 또는

1.8%의 미스율을 보여 줌으로 우수한 성능을 보여준다.



(그림 5) 전체 쓰기 요청에 따른 실제 write-back 비율

(그림 5)는 제안된 캐시를 사용하는 경우 발생하는 전체 쓰기 요청에 대한 실제 write-back이 발생한 비율을 보여준다. 여기서 x축의 AVG1과 AVG2는 전체 평균값과 4K 캐시를 제외한 평균값을 각각 나타낸다. Write-back 발생은 제안된 캐시를 사용할 경우 발생하는 과부하로서 이 비율이 높은 경우 제안된 캐시의 전체 성능을 저하 시킬 수 있다. 그러나 제안하는 캐시에서의 실제 write-back이 발생하는 횟수의 전체 평균 비율은 25%였으며, 4K 캐시를 제외한 평균은 약 20%로서 캐시 사이즈가 증가함에 따라 해당 비율은 감소하여 전체 성능에는 큰 영향을 주지 않는다.

<표 1> 속도 성능 측정

캐시 크기	미스	write-back	실제 접근	총접근 요청	속도증가비율
4	23,722	11,250	34,972	62,244	44%
12	16,032	7,058	23,090	62,244	63%
20	12,870	5,100	17,970	62,244	71%
28	11,485	4,479	15,964	62,244	74%
40	10,320	3,967	14,287	62,244	77%

위의 표는 제안 캐시를 사용하는 경우 얻을 수 있는 속도 성능향상을 데이터 요청에 따른 테이블 접근 횟수 기반 상대적 비율로 나타낸다. 비교 모델은 캐시를 채택하지 않는 모델로서 매핑 테이블을 플래시 메모리의 여유 공간에 저장을 한다. 이 경우 쓰기 연산은 테이블 접근시간이 필요 없게 되며 읽기 연산은 검색 작업을 거쳐야 한다. 따라서 비교 대상의 동작 조건은 한 번의 검색으로 부가적인 과부하 없이 접근요청이 처리된다. 반변 제안 캐시는 미스인 경우와 write-back이 발생한 경우만 실제 접근이 발생한다. <표 1>의 속도 증강 비율은 총 접근요청에서 실제 접근이 발생한 비율로 제안 캐시가 상대적으로 작은 횟수의 접근을 함으로 얻는 상대적 속도 성능 향상을 보여준다. 해당 성능 향상 비율은 캐시 사이즈가 증가함에 따라 미스율이 감소하여 증가한다.

5. 결론

플래시 메모리의 대용량화는 주소 매핑 테이블의 크기를 증가시키는 문제점과 독립적인 맵 블록을 유지할 경우 추가적인 접근 시간이 소요되는 단점이 있다. 본 논문에서는 하이브리드 변환 기법 기반의 플래시 메모리 파일 시스템에서 페이지 주소 캐시를 이용한 효율적인 주소 테이블 관리 정책을 제안하였다. 제안한 페이지 주소 캐시는 다양한 메타 데이터 기반의 전체 테이블의 정보를 맵 블록을 이용하여 효율적으로 통합 관리함으로써 PC 벤치마크에서 평균 1.2% 이하 그리고 generaluse에서 평균 1.8% 이하의 우수한 성능을 보였다. 또한 전체 쓰기 연산 요청에서 실제 쓰기 연산의 낮은 실행으로 전체 쓰기 연산에서 발생하는 과부하를 줄여 주었다.

참 고 문 헌

[1] Understanding the Flash Translation Layer (FTL) Specification, Intel Corporation, 1998.

[2] E. Harari, R. D. Norman, and S. Mehrotra, “Flash EEPROM System”, US Patent, No. 5,602,987, Dec. 1993.

[3] A. Ban, “Flash File System Optimized for Page-mode Flash Technologies”, US Patent, No. 5,937,425, Oct. 1997.

[4] A. Ban, “Flash File System”, U. S. Patent 5,404,485, 1995.

[5] T. Shinohara, "Flash Memory Card with Block Memory Address Arrangement," United States Patent, No. 5,905,993, 1999.

[6] J. W. Park, S. H. Park, G. H. Park, and S. D. Kim,

"An integrated mapping table for hybrid FTL with fault-tolerant address cache", IEICE Electronics Express, Vol.6 No.7, pp368-374, April, 2009.

[7] J. Kim, J. M. Kim, S. H. Noh, S. L. Min, and Y. Cho, "A Space-Efficient Flash Translation Layer for Compact Flash System," IEEE Transactions on Consumer Electronics, Vol.48, No.2, pp. 366-375, 2002.

[8] S. W. Lee, D. J. Park, T. S. Chung, D. H. Lee, S. Park, and H. J. Song, "A Log Buffer-Based Flash Translation Layer Using Fully-Associative Sector Translation," ACM Transactions on Embedded Computing Systems, Vol.6, No.3, Article 18, 2007.

[9] J. U. Kang, H. Jo, J. S. Kim, and J. Lee "A Superblock-based Flash Translation Layer for NAND Flash Memory" Proc. of EMSOFT, pp 161-170, 2006.

[10] http://www.futuremark.com/products/pcmark05/

[11] S. H. Park, J. W. Park, J. M. Jeong, J. H. Kim, and S. D. Kim, "A Mixed Flash Translation Layer Structure for SLC-MLC Combined Flash Memory System", Proc of IEEE Workshop on Storage and I/O Virtualization, Performance, Energy, Evaluation and Dependability 2008.

김 정 길

1987년 : Univ. of Redlands, CA, U. S. A. B.S. in Computer Science
2003년 : 연세대학교 대학원 (공학석사)
2006년 : 연세대학교 대학원 (공학박사)
2006년～2007년 : 연세대학교 컴퓨터과학과 PostDoc,
2007년～2008년 : 연세대학교 컴퓨터과학과 연구교수
2008년～현 재 : 남서울대학교 컴퓨터학과 조교수
관심분야 : 컴퓨터 구조, 멀티미디어 임베디드 시스템, RFID 미들웨어, 3D 콘텐츠