논문 2011-06-34

시뮬링크를 이용한 플래시메모리 저장장치 성능 모델링

(Performane Modeling of Flash Memory Storage Systems Using Simulink)

민 항 준, 박 정 수, 이 주 일, 민 상 렬, 김 강 희*
(Hang Jun Min, Jeong Su Park, Joo Il Lee, Sang Lyul Min, Kanghee Kim)

Abstract : The complexity of flash memory based storage systems is high due to diverse host interfaces and other design choices such as mapping granularity, flash memory controller execution models and so on. Thus, it is possible that the actual performance after implementation is not consistent with the target performance. This paper demonstrates that the performance prediction of flash memory based storage systems is possible through performance modeling that takes into account various design parameters. In the performance modeling, the FTL, which is the core element of flash memory based storage systems, is modeled as a set of (copy-on-write) logs and their interactions. Also, the flash memory controller is modeled based on the classification proposed in the design of the Ozone flash controller. In this study, the performance model has been implemented using Simulink and experimental results are presented and analyzed.

Keywords : Flash memory, Performance modeling, Storage system, Embedded system, Simulink

I. 서 론

플래시메모리는 임의 접근 속도가 빠르고 전력 소모가 적으며, 진동 및 충격에 강한 특성 등으로 인해 이동형 기기, 일반 PC및 서버 등 다양한 분야에서 사용되고 있다. 이를 활용한 저장장치들은 작은 크기의 카드 제품에서부터 기존의 하드 디스크의 크기를 갖는 Solid State Drive (SSD)까지 매우 다양한 인터페이스를 지원하고 있으며, 각 인터페이스에서 요구하는 대역폭과 용량에 따라 다양한 구조를 가진다.

* 교신저자(Corresponding Author)
논문접수 : 2011. 02. 28., 수정일 : 2011. 03. 23.,
채택확정 : 2011. 04. 11.
민항준, 박정수, 이주일, 민상렬 : 서울대학교 전기컴퓨터공학부
김강희 : 숭실대학교 정보통신전자공학부
※ 이 논문은 2010년도 정부(교육과학기술부)의 재원으로 한국연구재단(No. 2010-0015149) 및 삼성전자 반도체사업부의 지원을 받아 수행된 연구임.

플래시메모리 저장장치는 소프트웨어 계층인 FTL (Flash Translation Layer) [1-3]과 하드웨어로 구성되는 플래시메모리 제어기 및 다수 개의 플래시메모리 칩으로 구성된다. FTL은 하드디스크와 동일한 블록 장치 인터페이스를 제공하는 소프트웨어 계층으로서 주소 재 사상 정보를 이용하여 호스트 시스템의 요청을 플래시메모리 연산으로 변환하는 기능을 담당한다. 플래시메모리 제어기는 FTL에서 변환된 플래시메모리 연산을 플래시메모리 칩으로 발행하여 처리하는 역할을 담당한다.

본 논문에서는 FTL이 수행하는 주소 재 사상 기법을 다수개의 로그 (Log)들과 그들의 상호 작용으로 모델링 하였다. 플래시메모리 제어기는 플래시메모리 제어기 구조에 대한 기존 연구인 Ozone [7]에서 제시한 순차 (Sequential), 교차 (Decoupled) 및 비순차 (Out-of-order) 연산 처리 기법에 따라 모델링 하였다. 또한 정교한 모델링을 위하여 플래시메모리 연산 지연 시간 및 데이터 전송 속도와 같은 플래시메모리 자체의 특성과 플래시메모리 칩 및 버스의 개수 등의 구성을 매개 변수화 하였으며, 이들의 활용을 위해 본 논문에서 모델링한 모델들을 매스웍스사의 시뮬링크를 이용하

여 구현하였다. 이로 인해 실제 플래시메모리 저장장치를 구현하지 않고도 플래시메모리 저장장치의 성능을 빠르게 예측할 수 있다. 또한 본 논문에서 제작된 플래시메모리 저장장치 모델들은 라이브러리 및 컴포넌트의 형태로 제작되어 추후 유지 보수 및 새로운 응용에의 활용이 용이하게 하였다.

본 논문의 구성은 다음과 같다. 2장에서는 플래시메모리와 플래시메모리 저장장치의 중요 컴포넌트들에 대한 배경지식을 설명하고 3장에서 본 논문에서 수행한 성능 모델링에 대하여 기술한다. 4장에서는 본 논문에서 구현한 성능 모델의 동작 결과 및 활용 예를 제시하고, 마지막 5장에서는 향후 과제와 함께 결론을 맺는다.

II. 배경 지식

본 장에서는 플래시메모리 고유의 특성과 FTL 및 플래시메모리 제어기의 역할, 그리고 플래시메모리 저장장치 시뮬레이터에 대한 기존 연구들에 대해 간략하게 살펴본다. 또한 본 논문에서 모델링한 결과를 구현하기 위해 사용한 매스웍스사의 시뮬링크에 대해서도 간략히 소개한다.

1. 플래시메모리의 특성

플래시메모리는 동작 방식에 있어서 기존 저장매체와는 다른 특성을 지닌다. 플래시메모리는 HDD와 같은 기존의 저장 매체와는 달리 덮어쓰기를 허용하지 않으며, 이를 위해서는 소거 연산이 필요하다. 즉, 동일한 페이지에 쓰기 연산이 반복적으로 수행되어야 한다면 해당 페이지는 반드시 소거 연산이 선행되어야 한다. 또한 플래시메모리에는 쓰기 및 소거 횟수 제한이 있다. 하나의 셀에 하나의 비트 정보만 저장하는 SLC의 경우에는 약 10^5 회 이내로 제한되어 있는 반면, 하나의 셀에 여러 비트 정보를 저장하는 MLC의 경우에는 약 10^3 회 이내로 제한된다. 최근 플래시메모리의 집적도가 높아지면서 쓰기 및 소거 횟수가 점차 감소되는 추세이다.

[그림 1]은 플래시메모리의 읽기/쓰기/소거 연산 과정을 나타낸다. 읽기 연산의 경우, 해당 요청이 플래시메모리에 전달되면 일정한 읽기 지연 시간을 소모하며 해당 페이지의 정보를 플래시메모리 내부의 페이지 버퍼로 옮기는 작업을 수행한다. 읽기 지연 시간이 흐른 후 플래시메모리 칩은 '준비(Ready)' 상태가 되고 이 후에 데이터를 전송함으로써 읽기 연산이 완료된다.

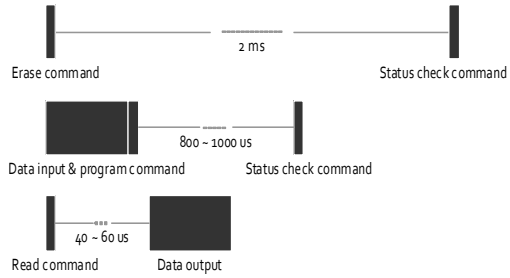

그림 1. 플래시메모리의 읽기/쓰기/소거 연산
Fig. 1. Flash read/write/erase operation

쓰기 연산의 경우, 쓰기 요청과 함께 데이터가 플래시메모리의 페이지 버퍼에 전달되면 일정한 쓰기 지연 시간을 소모하며 해당 페이지로 페이지 버퍼에 입력된 데이터를 쓰는 작업을 수행한다. 쓰기 지연 시간이 흐른 후 플래시메모리 칩은 '상태(Status)' 명령을 전송함으로써 쓰기 연산이 완료된다. 소거 연산의 경우, 소거 요청이 플래시메모리에 전달되면 일정한 소거 지연 시간을 소모하며 해당 블록의 정보를 소거하게 된다. 소거 지연 시간이 흐른 후 플래시메모리 칩은 '상태 (Status)' 명령을 전송함으로써 소거 연산이 완료된다. 이처럼 플래시메모리 연산은 일정한 지연시간을 기준으로 연산 시작 요청과 완료 요청으로 분리된다.

2. Flash Translation Layer(FTL)

앞서 살펴본 바와 같이 플래시메모리는 고유의 특성으로 인해 기존 저장장치와는 다른 동작 특성을 지닌다. FTL은 소프트웨어 계층으로서, 앞서 살펴본 플래시메모리 고유의 특성을 감추어 읽기/쓰기의 인터페이스를 갖는 기존의 블록 디바이스 인터페이스와의 호환성을 보장한다. 이를 통해 기존의 소프트웨어 계층구조에 큰 변화를 주지 않으면서도 안정적인 동작을 보장하고, 전체적인 개발 시간을 단축한다. FTL이 수행하는 주요 기능에는 주소 재사상 및 무효 블록 처리 기법 등이 있다.

호스트로부터 발행되는 요청들은 일반적으로 시간적/공간적 지역성이 존재한다. 플래시메모리는 덮어 쓰기를 허용하지 않으므로 시간적/공간적 지역성을 갖는 쓰기 요청을 처리하기 위해서는 요청이 중복되는 블록에 대하여 소거 연산을 선행해야 한다. 이때, 정보의 소실을 방지하기 위해서는 소거가 진행되는 동안 기존의 유효한 페이지들을 읽어 다른 장소에 보관해 두었다가 이후 소거가 완료되면 새

로 들어온 요청과 조합하여 해당 블록에 다시 기록해 주어야 한다. 이로 인해 발생하는 오버헤드를 감추기 위해 FTL은 일반적으로 플래시메모리를 추가 쓰기 동작을 수행하는 로그(Log) [8]로 간주하여 관리한다. 플래시메모리를 로그의 형태로 관리하므로 호스트의 요청에 대한 주소 재 사상 기법과 시간적/공간적 지역성이 있는 요청으로 인해 무효화된 블록들을 재사용하기 위한 무효 블록 처리 기법이 필요하다.

FTL의 주소 재 사상 기법은 주소 관리의 단위에 따라 페이지 사상/블록 사상/하이브리드 사상 기법으로 분류된다. 페이지 사상 기법은 호스트의 요청을 페이지 단위로 관리하는 기법으로, 블록 사상 기법에 비해 페이지가 저장되는 위치를 자유롭게 선택할 수 있고 크기가 작은 요청에 대해 비교적 빠른 속도로 처리할 수 있는 장점이 있다. 하지만 저장장치의 용량이 늘어남에 따라 주소 재 사상 정보 등의 메타정보를 유지하기 위해 필요한 메모리 및 저장 공간이 급격하게 증가하는 단점이 있다.

블록 사상 기법은 호스트의 요청을 소거 연산의 단위인 블록 단위로 관리하는 기법이다. 페이지 사상 기법에 비해 메타정보를 유지하기 위한 메모리 및 저장 공간의 소모가 적고 구현이 비교적 쉬운 장점이 있는 반면, 작은 크기의 호스트 요청에 의한 응답이 페이지 사상 기법에 비해 늦어진다는 단점이 있다. 이는 호스트에서 들어온 요청의 크기가 블록보다 작은 경우, 기존의 블록에서 유효한 페이지들을 병합하는 과정이 필요하기 때문이다.

하이브리드 사상 기법은 기존의 페이지 사상 기법과 블록 사상 기법의 문제점을 개선하기 위해 고안된 기법이다. 하이브리드 사상 기법은 기본적으로 블록 사상 기법으로 동작하며, 작은 크기의 호스트 요청이 들어올 경우에는 페이지 사상 기법을 적용하는 방식으로 동작한다. 하이브리드 사상 기법은 두 가지 방식의 장점을 취할 수 있지만, 추후에 무효 블록 처리나 병합을 실행할 때 페이지 단위로 관리되는 영역의 병합 손실이 크며 구현이 상대적으로 복잡해지는 단점이 있다. 해당 기법으로는 BAST, FAST, LAST, SAST, KAST [1-3, 9-10] 방식 등의 다양한 연구들이 진행되어 오고 있다.

이외에도 FTL에서는 일정한 양의 블록을 별도로 보관해 두고, 불량 블록이 발생하는 경우 이들과 교체하여 안정적인 동작을 보장한다. 또한 기존 자료에 대한 덮어쓰기 및 병합동작 등으로 인해 무효화된 영역을 재사용하기 위한 다양한 무효 블록 처리 기법 [12]들을 적용한다.

3. 플래시메모리 연산 처리 기법

플래시메모리의 연산은 앞서 살펴본 바와 같이 연산 시작 요청과 연산 완료 요청으로 구분할 수 있다. 플래시메모리 제어기는 각 연산에서 필요한 지연시간을 최대한 감추고 여러 칩에서 다수의 연산이 병렬적으로 수행되게 함으로서 연산 처리량(Throughput)을 높인다. 이를 위해 플래시메모리 제어기에서는 다양한 연산 처리 기법을 사용한다. 본 장에서는 플래시메모리 제어기 구조에 대한 기존 연구인 Ozone[7]에서 제시한 순차 실행 기법, 교차 실행 기법, 비순차 실행 기법에 대하여 살펴본다.

3.1 플래시메모리 제어기의 순차 실행 기법

순차 실행 기법은 시작/완료 요청으로 분리된 플래시메모리의 연산을 들어온 순서대로 처리하는 것을 말한다. 예를 들어 두 개의 플래시메모리 연산이 서로 다른 플래시 칩에서 처리되어야 할 때, 나중에 도착한 연산은 버스와 칩이 사용 가능하더라도 먼저 발행된 연산이 완료될 때까지 기다려야 한다. 순차 실행 기법은 동작이 간단하므로 제어기의 구성을 단순하게 한다는 장점이 있는 반면 플래시메모리의 병렬성을 전혀 활용하지 못한다는 단점이 있다.

3.2 플래시메모리 제어기의 교차 실행 기법

교차 실행 기법은 시작/완료 요청으로 분리된 플래시메모리의 연산이 동일한 버스 내에 존재하는 칩들 사이에서 중첩되어 실행되도록 하는 기법이다. 이를 통해 하나의 버스에 다수 개의 칩이 연결되어 있는 경우 이들을 병렬적으로 동작시켜 성능을 향상시킬 수 있다.

교차 실행 기법에는 각각의 플래시메모리 연산에 대한 처리 순서를 지켜야 하는 제약이 존재한다. 예를 들어 서로 다른 칩으로 발행되는 두 개의 플래시메모리 연산이 있고 처음 발행되는 연산이 뒤이어 발행되는 연산보다 지연 시간이 훨씬 긴 연산인 경우에는 뒤이어 발행된 연산이 먼저 수행이 완료가 될 수 있더라도 처음 발행된 연산이 완료될 때까지 기다려야 한다.

3.3 플래시메모리 제어기의 비순차 실행 기법

비순차 실행 기법은 교차 실행 기법의 제약사항을 극복하여 연산 처리 성능 및 처리량을 극대화하기 위하여 고안되었다. 비순차 실행 기법은 기존의 수퍼스케일러 (Superscalar) 프로세서에서 아이디어를 얻어 교차 실행 기법에서 뒤이어 발행된 연산이 먼저 완료될 수 있도록 허용함으로써 성능을 극대화한다.

플래시메모리 연산은 동일한 칩에서 수행되어야 하는 연산들에 한해서만 데이터 의존관계가 존재하기 때문에 이러한 데이터 의존관계 이외에는 비순차적으로 플래시메모리 연산을 수행하는 것이 가능하다. 비순차 실행 기법에서는 동일한 칩에서 수행되어야 하는 플래시메모리 연산에 대해서는 수행 순서를 유지하는 반면, 다른 칩에서 수행되는 플래시메모리 연산에 대해서는 수행 순서를 유지하지 않는다. 이를 통해 플래시메모리의 병렬성을 최대로 활용하고, 각각의 플래시메모리 연산의 지연 시간을 최대한 감출 수 있다.

4. 플래시메모리 저장장치 시뮬레이터

기존 연구에서 플래시메모리 저장장치의 성능을 예측하기 위해 개발된 시뮬레이터로서 대표적인 것은 마이크로 소프트에서 개발한 SSDSim [4]이다. SSDSim은 CMU의 PDL에서 배포한 DiskSim [5]을 기반으로 작성된 SSD전용 시뮬레이터로서, FTL에 특화된 마모 평준화 기법과 무효 블록 처리 기법, 그리고 플래시메모리 연산 처리 기법 중 하나인 인터리빙 (Interleaving)에 의한 성능 변화를 확인하기 위해 개발되었다. 또 다른 시뮬레이터로는 Penn Univ.의 FlashSim [6]이 있다. FlashSim은 FTL의 주소사상기법 변화에 따른 SSD 성능 및 전력소모량을 예측하기 위하여 개발되었다.

기존의 시뮬레이터들은 다양한 플래시메모리 저장장치의 성능결정 인자들 중 제한된 일부 인자들만 변경이 가능하도록 허용하고 있으며, 실제 플래시메모리 저장장치의 모든 성능 결정 인자를 반영하지는 않고 있다. 따라서 플래시메모리 저장장치의 다양한 설계 영역들을 모두 고려하기에는 한계가 있다.

Ⅲ. 성능 모델링

본 장에서는 플래시메모리 저장장치의 구조를 살펴보고, 각각의 요소에 대한 성능 모델링에 대해 자세히 살펴본다.

1. 플래시메모리 저장장치 전체 모델 구성

일반적으로 플래시메모리 저장장치는 세 부분으로 구성된다. (1) 호스트와의 인터페이스를 담당하는 호스트 인터페이스 부, (2) 호스트의 요청을 플래시메모리 연산으로 변환하며 주소 재 사상 및 무효 블록 처리 등의 전체 저장장치의 관리를 담당하는 FTL, (3) 플래시메모리 연산을 스케줄링 하여 실제 플래시메모리로 발행하는 역할을 하는 플래시메모리 제어기 및 플래시메모리 칩이 그것이다.

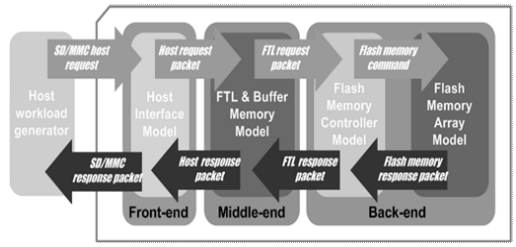

그림 2. 플래시메모리 저장장치의 구조
Fig. 2. Structure of the flash memory storage

[그림 2]는 플래시메모리 저장장치의 전체 구조를 나타낸다. 본 논문에서는 위와 같이 플래시메모리 저장장치 모델을 (1) 워크로드 생성기 모델, (2) 호스트 인터페이스 모델, (3) FTL 및 버퍼 모델, (4) 플래시메모리 제어기 모델로 크게 네 가지 요소로 구분하고, 각 모델들을 통합하여 실제 플래시메모리 저장장치에 대한 성능 모델링을 수행하였다. 각 모델에 대한 자세한 설명은 다음 장을 참고한다.

2. 요소 모델

본 장에서는 앞서 기술된 플래시메모리 저장장치의 요소 모델들을 각각 살펴본다.

2.1 워크로드 생성기 모델

워크로드 생성기 모델은 호스트 시스템을 모델링하며, 플래시메모리 저장장치 모델에 대한 호스트의 I/O요청을 제공한다. 호스트 시스템에서의 I/O요청 패턴은 매우 다양하다. 이를 반영하기 위해 본 논문의 워크로드 생성기 모델은 실제 워크로드 트레이스를 재생할 뿐 아니라 합성 워크로드 또한 제공한다. 합성 워크로드는 총 5가지의 I/O 패턴의 생성이 가능하다. 제공하는 패턴은 다음과 같다.

1) Uniform: 모든 주소 영역 참조의 확률이 동일
2) Hot/Cold : 공간적 지역성을 반영
3) LRU : 시간적 지역성을 반영
4) Hot/Cold + LRU : 3)과 4)의 조합
5) Sequential : 순차적인 주소 접근

[그림 3]은 시뮬링크를 통해 구현된 워크로드 생성기 모델의 내부이다. Simevent 패키지에 포함된 Entity generator, Server, Gate 등의 기본 라이브러리를 조합하고, S-function을 통해 Discrete

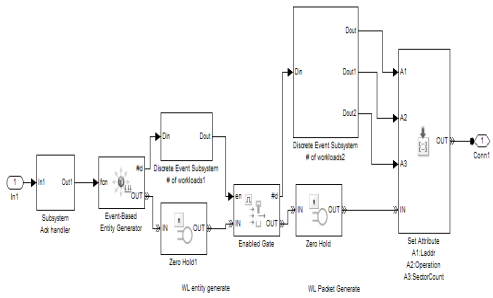

그림 3. 워크로드 생성기 모델
Fig. 3. Workload generator model

Event Subsystem 모듈 안에 C 코드를 통합하는 방식으로 원하는 워크로드를 생성할 수 있도록 하였다. 워크로드를 결정하는 파라미터는 워크스페이스에서 정의된 변수를 통해 전달된다.

2.2 호스트 인터페이스 모델

플래시메모리 저장장치의 호스트 인터페이스로는 SD/MMC, USB, SATA/SAS, PCI-Express 등의 다양한 형태가 존재한다. 호스트 인터페이스가 달라지면 데이터 전송을 위한 프로토콜과 대역폭 등이 서로 달라지므로, 호스트 인터페이스 모델은 각 인터페이스별 커맨드의 처리 시간과 데이터 전송 시간을 반영하도록 모델링 되어야 한다. 본 논문에서는 호스트 인터페이스의 대역폭과 커맨드 처리 시간을 매개 변수화 하여 이를 반영하였다. [그림 4]는 시뮬링크의 Simevent 패키지 라이브러리의 조합으로 구현한 호스트 인터페이스 모델의 내부이다.

2.3 FTL 및 버퍼 모델

FTL은 덮어쓰기를 허용하지 않는 플래시메모리의 특성상 주로 기존 LFS [8]에서 제시한 로그의 형식으로 플래시메모리를 관리한다. 플래시메모리를 로그로 관리하기 위해서는 주소 재 사상과 무효 블록 처리를 위한 메타정보의 관리가 필요하며, 이러한 메타정보들도 플래시메모리에 로그의 형태로 저장이 되어 추후에 전원공급이 중단되더라도 다시 사용 가능한 형태로 관리가 되어야 한다. 본 논문에서는 FTL이 유한개의 스트림 (Stream)을 갖는 Copy-On-Write (COW) 방식의 로그들의 집합으로 모델링 하였다. 여기서 COW 방식이란 기존의 데이터가 저장된 공간이 아닌 다른 빈 공간에 복사본의 형태로 저장하는 방식을 의미한다. FTL을 유한개의 스트림을 갖는 로그들의 집합으로 간주하는 이유는, FTL의 주된 작업인 주소 재 사상과 무효 블록 처리를 위한 메타데이터들 역시 별도의 로그의 형태로 관리되기 때문이다.

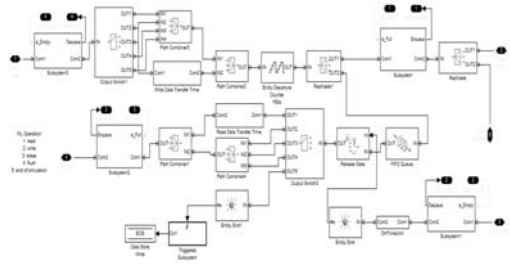

그림 4. 호스트 인터페이스 모델
Fig. 4. Host interface model

이를 확장하면, FTL은 데이터 저장용의 D-로그, 사상 정보 보관용의 M-로그, 데이터 유효성 정보 보관용의 L-로그, 비 휘발성 버퍼용 W-로그, 그리고 전원 오류 복구용의 C-로그 와 같이 총 5가지 타입의 로그로 구성될 수 있다. 각 로그에 대한 설명은 다음과 같다.

1) D-로그 : 호스트로부터 전달된 사용자 데이터를 저장하기 위해 할당된 로그이다. 데이터 자체만을 저장하므로 로그 중에서는 가장 큰 용량을 차지하게 된다.

2) M-로그 : D-로그에서 데이터가 저장되는 공간의 주소 재 사상 정보를 저장하기 위해 할당된 로그이다. M-로그 자체도 로그의 형태로 관리되기 때문에 M-로그의 주소 재 사상 정보를 저장하기 위한 또 다른 M-로그가 존재할 수 있다. 이러한 M-로그는 모든 사상정보가 한 페이지 내에 저장될 때까지 계층적으로 생성된다.

3) L-로그 : 각 로그에서 갱신된 영역에 대한 무효 정보를 보관하기 위해 할당된 로그이다. 각 로그의 무효 정보는 추후에 무효 블록 처리를 위해 사용된다.

4) W-로그 : 하이브리드 사상 기법에서 크기가 작고 임의로 사상되는 호스트의 요청을 위해 사용되는 비휘발성 버퍼용 로그이다. W-로그를 통해 다양한 사상 관리 단위를 갖는 FTL의 구현이 가능하다.

5) C-로그 : 전원 오류 복구를 위해 사용되는 로그이다. 만약 플래시메모리 저장장치가 동작 중에 전원 이상으로 초기화가 되었다면 추후에 전원 오류 복구를 수행하게 된다. 이때 전원 오류 복구를 위해서 관리하는 모든 영역을 확인하는 것은 소모적인 일이므로, 이를 방지하기 위해 주기적으로 체크포인트를 수행한다. 이러한 체크포인트 정보를 보관하기 위해 사용된다.

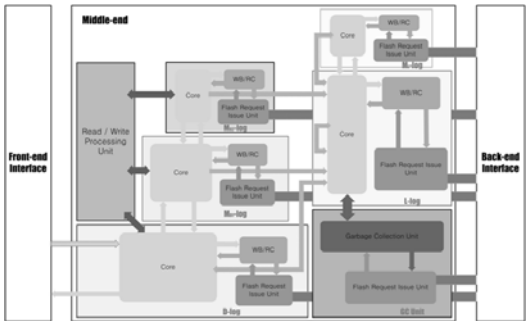

그림 5. 로그를 이용한 FTL 모델링 예시
Fig. 5. An example of the FTL modeling using logs

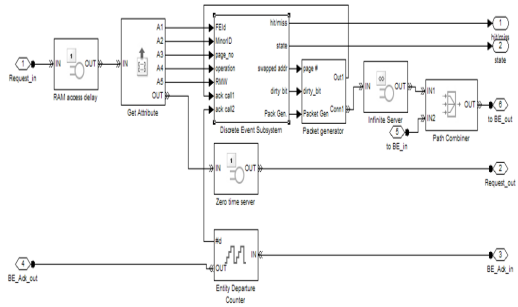

그림 6. 휘발성 버퍼 모델
Fig. 6. Volatile buffer model

[그림 5]는 위에서 제시한 각 로그를 이용하여 구성한 FTL의 하나의 예이다. 하나의 D-로그와 D-로그를 위한 두 개의 M-로그 그리고 통합 무효 정보 저장을 위한 하나의 L-로그와 L-로그의 주소 재 사상 정보를 관리하는 M-로그로서 전체 FTL이 구성된다. 여기에 추가로 무효 블록 처리 유닛이 있어서 각 로그의 무효 블록을 수집 및 재활용하는 역할을 담당한다.

이렇게 구성된 FTL의 동작은 다음과 같다. D-로그는 호스트의 읽기/쓰기 요청을 직접적으로 처리하며, 이 과정에서 요구되는 메타정보에 대한 읽기/쓰기 요청은 M-로그(주소 재 사상 정보) 와 L-로그 (무효화 정보)로 전달된다. 각 로그는 각자 고유의 쓰기 및 읽기용 휘발성 버퍼를 소유하고 있으며, 이를 이용하여 응답시간을 최적화 한다. 버퍼에 해당 요청에 대한 내용이 없다면 적절한 버퍼 교체 정책에 의거하여 플래시메모리 연산을 생성하고, 이를 플래시메모리 제어기로 전송한다. 이러한 동작은 각 M-로그와 L-로그에서도 동일하게 진행되며, 계층구조에 따라 가장 높은 곳에 위치한 로그의 경우 페이지 하나에 대한 요청만이 들어오기 때문에 항상 버퍼에서 처리할 수 있다. L-로그에서는 각 로그에 대한 무효 블록의 개수를 모니터링 하면서, 위 값이 임계값을 넘어서게 되면 무효 블록 처리 유닛이 동작하여 무효 블록들의 수거 및 재활용을 실시한다.

위에서 제시한 방법에 따라 FTL을 구성하는 경우, 실제로 사용되는 플래시메모리 용량에 따라 계층적 구조의 단계는 달라진다. 단계가 높아짐에 따라 더 많은 로그가 사용되는데, 이때 다양한 로그들의 연결을 용이하게 수행할 수 있도록 각 로그를 컴포넌트화 하고, 로그간의 요청 전달은 단일화된 패킷 기반의 인터페이스를 사용하였다. 또한 위에서 기술한 바와 같이 각 로그는 휘발성 쓰기 및 읽기 버퍼를 사용함으로써 성능을 향상 시킬 수 있는데, 각 로그의 버퍼 교체 정책은 최근성 (Recency)을 반영하는 버퍼 관리 정책인 LRU방식으로 구현되어 있으며, 버퍼의 크기 및 교체 정책은 사용자의 필요에 따라 적절하게 변경될 수 있도록 매개 변수화 하였다.

[그림 6]은 로그의 휘발성 버퍼를 시뮬링크로 구현한 모습이다. 로그로 전달되는 메시지의 이동 과정은 Simevent 패키지의 라이브러리를 활용하였으며, 버퍼교체 정책은 C언어로 구현하여 S-function을 통해 통합하였다.

페이지 사상, 블록 사상, 하이브리드 사상 기법 등의 주소 사상 기법은 D-로그의 버퍼 메모리 관리 단위를 조정하여 반영할 수 있다. D-로그의 버퍼 메모리 관리 단위를 페이지로 하는 경우 페이지 사상 기법, 블록단위로 하는 경우 블록 사상 기법, 그리고 비 휘발성 쓰기 버퍼인 W-로그를 활용하는 경우 하이브리드 사상 기법으로 각각 모델링할 수 있다.

2.4 플래시메모리 제어기 모델

플래시메모리 제어기는 다수 개의 플래시메모리 버스와 다수 개의 칩을 관리하며 FTL에 의해 변환된 플래시메모리의 연산을 처리하는 역할을 담당한다.

[그림 7]은 플래시메모리 제어기의 구성을 나타내며, 플래시메모리 제어기의 구성 및 동작에 대한 자세한 내용은 기존의 Ozone [7]에서 연구된 사항을 참고하였다. 이에 관한 모델링 및 구현은 기존 연구 [11]에서 시뮬링크를 이용하여 수행되었으며, 본 논문에서는 상기 결과물을 통합 모델의 주요 모듈로 활용하였다.

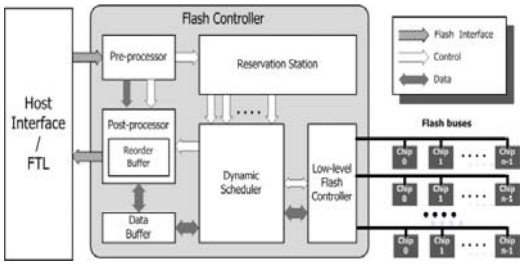

그림 7. 플래시메모리 제어기 구성도 [7]
Fig. 7. Schematic of the flash memory controller

3. 통합모델 구현

앞서 살펴본 네 가지의 요소 모델들은 모두 매스웍스사의 시뮬링크를 이용하여 구현되었다. [그림 8]은 시뮬링크를 이용하여 각 요소 모델을 구현하고 이를 통합한 모습이다. 추가로 구현된 부분은 전체 모델에서 사용하는 매개 변수 값을 설정하는 부분과 모델링을 통한 결과를 시각적으로 보여주기 위한 부분이다. 이렇게 구성된 통합 모델은 실제 플래시메모리 저장장치 구현에 들어가기에 앞서 다양한 설계 인자 변경에 따른 성능 예측을 위한 용도로 사용할 수 있다. 이를 이용한 활용예는 다음 장에서 설명한다.

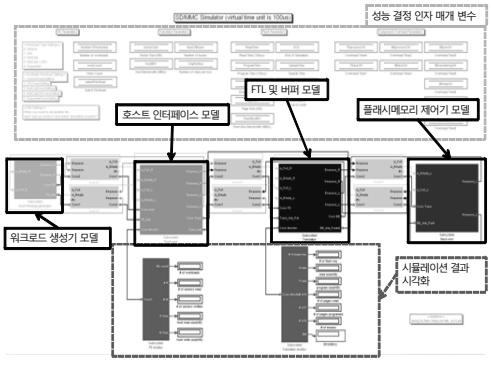

그림 8. 시뮬링크 통합 모델 구현
Fig. 8. Simulink implementation of the intergrated model

Ⅳ. 모델링 활용예

최근 플래시메모리 시장에는 ONFI. HLNAND, Toggle mode NAND 등의 다양한 제품들이 출시되고 있다. 이들 제품들은 데이터 전송 대역폭 및 연산 지연 시간 등의 특성이 모두 다르기 때문에

표 1. 통합 모델 파라미터 설정
Table 1. Integrated model parameter setting

	설정 항목	설정값
정적 파라미터	플래시메모리 저장 장치 용량	16 GB
	호스트인터페이스 대역폭	150 MB/s
	FTL 사상 기법	페이지 사상 기법
	FTL 버퍼 관리 기법	LRU 방식
	FTL 폐 블록 회수 기법	Greedy 방식
	플래시메모리 연결 구성	버스 2개, 버스 당 2개의 칩
동적 파라미터	워크로드	Uniform, LRU, HCLRU
	플래시 연산 처리 기법	순차 (Sequential), 비순차 (Out-of-order)
	플래시메모리 버스 대역폭	40 MB/s, 133MB/s
	플래시메모리 페이지 크기	2 KB ~ 8 KB
	플래시메모리 연산 지연 시간	읽기 : 20 us ~ 60 us, 쓰기 : 200 us ~ 800 us, 지우기 : 2000 us
	FTL 버퍼 크기	512 KB ~ 4 MB

표 2. 자료 전송 대역폭 및 연산 지연시간 비교
Table 2. Comparison of data throughput and operation latencies

플래시메모리 타입	대역폭(MB/s)
Conventional 플래시 버스	40 MB/s

플래시메모리 타입	연산 지연 시간			페이지 크기
	읽기	쓰기	지우기	
SLC (Single Level Cell)	20 us	200 us	2000 us	2 KB
MLC (Multi Level Cell)	60 us	800 us	2000 us	8 KB

플래시메모리 저장장치를 설계할 때 어떤 플래시메모리를 사용하는 것이 원하는 제품 사양을 만족시킬 수 있는지 미리 예측해야 한다. 예를 들어 Conventional한 플래시메모리의 경우 기존의 제품들과 호환성 측면에서는 우수하지만 대역폭이 ONFI에 비해 낮다는 단점이 있다. 또한 SLC 타입의 NAND의 경우 MLC 타입에 비해 연산 지연 시간은 짧지만 단위 용량당 비용이 높다는 단점이 있다. 이러한 요소들을 모두 고려하여 최소한의 비용을 통해 제품의 스펙을 만족시킬 수 있는 솔루션을 제시하는 것은 설계 단계에서의 가장 큰 목표중의 하나이다.

본 논문에서는 통합 모델의 활용 예로써 ONFI 인터페이스를 지원하는 SLC 및 MLC 플래시메모리와 Conventional 인터페이스를 지원하는 SLC 및 MLC 플래시메모리를 비교하고, FTL에 사용되는 휘발성 버퍼의 용량에 따른 성능 차이를 실험해 보았다. 실험 환경 및 설정 값은 다음과 같다.

[표 1]은 통합 모델의 파라미터 설정 값을 나타낸다. 정적 파라미터는 모든 실험에서 동일하게 유지되는 설계 인자를 나타내며, 동적 파라미터는 실험 도중 변경되는 설계 인자를 나타낸다. [표 2]는

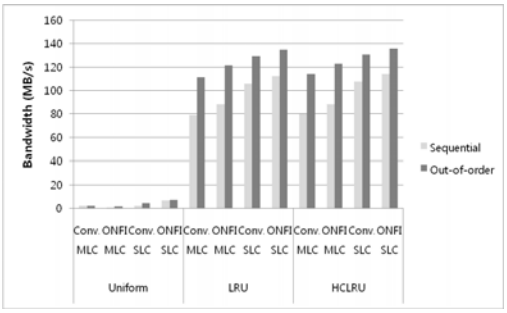

그림 9. 플래시메모리 특성 변화에 따른 성능 비교
Fig. 9. Performance comparison with varing flash memory characteristics

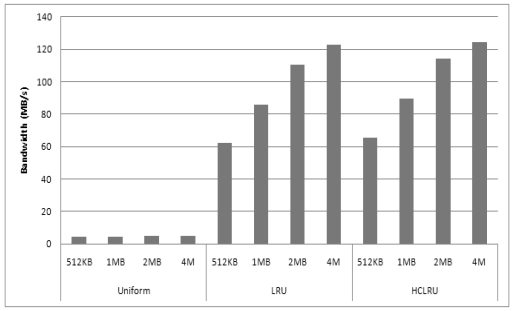

그림 10. FTL 휘발성 버퍼 용량에 따른 성능 비교
Fig. 10. Performance comparison with varing FTL volatile buffer size

Conventional 및 ONFI 인터페이스와 SLC 및 MLC 타입의 특성을 비교하고 있다. 대역폭은 ONFI 인터페이스가 Conventional 인터페이스보다 높고, 연산 지연 시간은 SLC 타입이 MLC 타입에 비해 짧다.

[그림 9]은 통합 모델을 활용하여 플래시 메모리 종류에 따른 성능 예측 결과를 그래프로 나타낸 것이다. 워크로드는 Uniform/LRU/HCLRU의 세 가지 타입을 사용하였으며, 플래시메모리 제어기는 순차/비순차로 처리하도록 설정하여 실험을 진행하였다. 위 결과에서 Uniform 워크로드는 타 워크로드에 비해 매우 낮은 성능을 보이고 있다. 이는 호스트 시스템의 요청이 지역성이 낮아 쓰기 버퍼 및 읽기 캐싱의 효과를 얻지 못하고 있음을 의미하는 것으로, 그 외에는 세 가지의 워크로드 모두 동일한 성능 비교 양상을 보이고 있다. 대역폭이 넓고 연산 지연 시간이 짧은 ONFI 인터페이스와 SLC 타입을 사용한 결과가 가장 좋은 성능을 보이는 반면, 대역폭이 낮고 연산 지연 시간이 높은 Conventional 인터페이스와 MLC 타입을 사용한 결과가 가장 낮은 성능을 보이고 있다. 이는 연산 처리 기법을 순차/비순차로 변경해도 동일하다.

위 실험에서 주목할 만한 부분은 ONFI 인터페이스에 MLC를 적용한 모델과 Conventional 인터페이스에 SLC를 적용한 모델의 비교 결과이다. 앞선 두 모델의 경우, 전자의 모델이 대역폭과 연산 지연시간 측면에서 모두 후자보다 우위를 점하고 있었기 때문에 성능이 높다는 것을 쉽게 예측할 수 있지만, 이와 같은 경우에는 상보적인 관계를 보이기 때문에 예측이 쉽지 않다. 이러한 경우 [그림 9]와 같이 본 논문에서 제시한 성능 모델을 통하여 [표 2]와 같은 상황에서는 Conventional SLC를 사용하는 것이 ONFI MLC를 사용하는 것보다 성능상 더 나은 선택임을 쉽게 도출해 낼 수 있다.

[그림 10]에서는 FTL에서 사용하는 휘발성 버퍼의 크기에 따른 성능 예측 결과를 그래프로 나타내었다. 실험 환경은 [표 2]의 Conventional SLC 타입을 가정하였다. 예상할 수 있는 바와 같이, 지역성이 없는 Uniform 워크로드의 경우 버퍼의 양을 늘려도 유의미한 변화를 이끌어 내지 못한 반면 지역성이 뚜렷하게 나타나는 LRU 및 HCLRU 워크로드에서는 휘발성 버퍼의 크기가 증가할수록 성능이 향상되는 경향을 확인할 수 있다. 지역성이 더 강한 HCLRU 워크로드의 경우 LRU 워크로드보다 성능 향상의 폭이 조금 높다. LRU 워크로드와 HCLRU 워크로드에서 버퍼의 크기를 1MB 이상 적용한 경우, 읽기 캐싱의 효과로 인해 플래시 버스 대역폭의 한계인 80MB/s를 넘어서는 성능 향상을 기대할 수 있다. 버퍼의 크기를 증가시킬수록 호스트 인터페이스 대역폭의 한계인 150MB/s에 근접해 지고 있지만, 버퍼 증가에 따른 성능 향상 폭이 점차 줄어드는 양상을 보이고 있으므로 이를 통해 원하는 성능과 버퍼 용량에 따른 가격 증가 사이의 trade-off 관계를 비교해 볼 수 있을 것이다.

V. 결론 및 향후 과제

본 논문에서는 플래시메모리 저장장치를 워크로드 생성기 모델, 호스트 인터페이스 모델, FTL 및 버퍼 모델, 플래시메모리 제어기 모델의 네 가지 요소로 정의하고 각각의 요소에 대한 모델링을 수행하였다. 각 모델들은 고유의 성능 결정인자들을 반영할 수 있도록 매개 변수화 하였으며, 패킷 형태의 단일화된 인터페이스를 지니는 컴포넌트 형태로 구

현하여 추후 확장 및 통합이 용이하도록 하였다. 본 논문에서 제시된 각 모델들은 매스웍스사에서 제공하는 시뮬링크로 구현되었으며, 이들은 하나의 모델로 통합되어 플래시메모리 저장장치의 성능 시뮬레이터로써 활용할 수 있다. 본문에서 제시된 활용예의 결과와 같이 이를 통해 실제 플래시메모리 저장장치를 구현하기에 앞서 다양한 설계 인자 변경에 따른 성능 예측을 손쉽게 수행할 수 있었다.

플래시메모리 제어기 모델의 검증은 기존 연구[11]에서 이루어 졌으나, 본 논문에서 제기된 FTL 및 버퍼 모델의 동작에 대한 검증은 아직 수행되지 않은 상황이다. 현재 다수개의 로그 형태로 구성되는 FTL에 대한 연구 및 구현이 진행되고 있는 상황이며, 추후 이를 활용하여 본 논문에서 제시된 FTL 및 버퍼 모델의 성능 검증을 수행할 예정이다.

참고문헌

[1] 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.18, 2007.

[2] J. U. Kang, H. Jo, J. S. Kim, and J. Lee, "A superblock-based flash translation layer for NAND flash memory", In Proceedings of the 6th ACM & IEEE International conference on Embedded Software (EMSOFT '06), pp. 161-170, 2006.

[3] J. S. Kim, J. M. Kim, S. H. Noh, S. L. Min, and Y. Cho, "A space-efficient flash translation layer for compactflash systems", IEEE Transactions on Consumer Electronics, Vol.48, Issue 2, pp. 366-375, 2002.

[4] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy, "Design tradeoffs for SSD performance", In Proceedings of USENIX Annual Technical Conference, pp. 57-70, 2008.

[5] J. S. Bucy, J. Schindler, S. W. Schlosser, and G. R. Ganger, "The DiskSim simulation environment version 4.0 reference manual", Technical report, CMU-PDL-08-101, Carnegie Mellon University, 2008.

[6] Y. Kim, B. Tauras, A. Gupta, and B. Urgaonkar, "FlashSim: A Simulator for NAND flash-based solid-state drives", Technical report, CSE 09-008, Penn State University, 2009.

[7] E. H. Nam, B. S. J. Kim, H. Eom, and S. L. Min, "Ozone (O3): An out-of-order flash memory controller architecture", IEEE Transactions on Computers, Vol. 60, No. 5, pp. 653-666, 2011.

[8] M. Rosenblum, and J. K. Ousterhout, "The design and implementation of a log-structured file system", In Proceedings of th thirteenth ACM Symposium on Operating Systems Principles (SOSP '91), pp. 1-15, 1991.

[9] H. Cho, D. Shin, and Y. Eom, "KAST: K-associative sector translation for NAND flash memory in real-time systems", IEEE DATE Conference, 2009.

[10] S. Y. Park, W. Cheon, Y. Lee, M. S. Jung, W. Cho, and H. Yoon, "A re-configurable FTL architecture for NAND flash-based applications", In Proceedings of International Workshop on Rapid System Prototyping, 2007.

[11] 민항준, 박정수, 이주일, 민상렬, "시뮬링크를 이용한 플래시메모리 제어기의 성능 모델링", 2010 대한임베디드공학회 추계학술대회 논문집, 2010.

[12] E. Gal, S. Toledo, "Algorithms and data structures for flash memories", ACM Computing Surveys, Vol.37, No.2, pp. 138-163, Jun. 2005.

저 자 소 개

민 항 준

2009년 : 성균관대학교 컴퓨터공학과 학사.
2011년 : 서울대학교 전기컴퓨터공학부 석사.
현재, LG Innotek 주임연구원.

관심분야 : 임베디드SW, OS, 플래시메모리, Bluetooth.
Email : fcsemi@gmail.com

박 정 수

2008년 : 서울대학교 전기공학부 학사.
2010년 : 서울대학교 전기컴퓨터공학부 석사.
현재, 서울대학교 전기컴퓨터공학부 박사과정.

관심분야 : 컴퓨터 구조, 임베디드 시스템, 플래시메모리.
Email : jspark@archi.snu.ac.kr

이 주 일

2010년 : 서울시립대 컴퓨터과학부 학사.
현재, 서울대학교 전기컴퓨터공학부 석사과정.

관심분야 : 컴퓨터 구조, 플래시메모리, 운영체제.
Email : jilee@archi.snu.ac.kr

민 상 렬

11983년 : 서울대학교 컴퓨터공학과 학사.
1985년 : 서울대학교 컴퓨터공학과 석사.
1989년 : University of Washington 전산학 박사.

1989~1990년 : IBM T.J Watson Reserch Center 객원연구원.
1990~1992년 : 부산대학교 컴퓨터공학과 조교수.
현재, 서울대학교 전기컴퓨터공학부 교수.
관심분야 : 컴퓨터구조, 병렬처리, 컴퓨터 성능측정.
Email : symin@snu.ac.kr

김 강 희

1996년 : 서울대학교 컴퓨터공학과 학사.
1998년 : 서울대학교 전기컴퓨터공학부 석사.
2004년 : 서울대학교 전기컴퓨터공학부 박사.

2004~2009년 : 삼성전자 무선사업부 책임연구원.
현재, 숭실대학교 정보통신전자공학부 조교수.
관심분야 : 실시간 시스템, 임베디드 시스템, 운영체제, 모바일 플랫폼.
Email : khkim@ssu.ac.kr