커널의 미리읽기를 고려한 압축파일시스템의 읽기성능향상

(Improving the Read Performance of Compressed File Systems Considering Kernel Read-ahead Mechanism)

안 성 용 † 현 승 환 ††
(Sungyong Ahn) (SeungHwan Hyun)
고 건 †††
(Kern Koh)

요 약 압축파일시스템은 비용효율성을 높이기 위해 모바일 장치 개발에 자주 사용된다. 그러나 압축파일시스템은 비압축파일시스템에 비해 읽기 성능이 떨어진다는 단점이 있다. 압축파일시스템의 읽기성능저하의 원인 중 하나는 커널 의 미리읽기 기법이다. 주된 이유는 압축파일시스템의 압축해제 오버헤드로 인해 미리읽기 미스패널티가 너무 크기 때문이다. 이 문제를 해결하기 위해 본 논문에서는 커널의 미리읽기 기법을 고려한 압축파일시스템의 읽기기법을 제안한다. 제안된 기법은 bulk read를 통해 저장장치의 성능을 향상시키는 동시에 선택적 압축해제를 통해 압축파일시스템의 압축해제 오버헤드를 줄인다. 우리는 리눅스 기반 시스템에서 널리 사용되는 압축파일시스템인 CramFS를 수정하여 제안된 기법을 구현하였으며 성능측정 실험을 통해 제안된 기법이 압축파일시스템의 메이저 페이지 폴트 처리 시간을 약 28%까지 단축시킬 수 있음을 보였다.

키워드 : 압축파일시스템, 운영체제, 미리읽기, 내장형시스템

Abstract Compressed filesystem is frequently used in the embedded system to increase cost efficiency. One of the drawbacks of compressed filesystem is low read performance. Moreover, read-ahead mechanism that improves the read throughput of storage device has negative effect on the read performance of compressed filesystem, increasing read latency. Main reason is that compressed filesystem has too big read-ahead miss penalty due to decompression overhead. To solve this problem, this paper proposes new read technique considering kernel read-ahead mechanism for compressed filesystem. Proposed technique improves read throughput of device by bulk read from device and reduces decompression overhead of compressed filesystem by selective decompression. We implement proposed technique by modifying CramFS and evaluate our implementation in the Linux kernel 2.6.21. Performance evaluation results show that proposed technique reduces the average major page fault handling latency by 28%.

Key words : Compressed Filesystem, Operating System, Read-ahead Mechanism, Embedded System

1. 서 론

오늘날 비용효율성은 모바일 장치 개발에 있어서 가장 중요한 요소이다[1]. CramFS[2,3]과 같은 압축파일시스템은 비용효율성을 높일 수 있는 매우 유용한 방법이다. 시스템개발자들은 압축파일시스템을 사용함으로써 모바일 장치의 제한된 저장공간을 좀 더 효율적으로 사용할 수 있다. 그러나 압축파일시스템은 비압축파일시스템에 비해 읽기성능이 나쁘다는 단점이 있다. 또한 우리는 대부분의 운영체제에서 사용하는 미리읽기 기법이 압축파일시스템의 읽기 성능을 더욱 악화시킬 수 있다는 점을 주목해야 한다.

미리읽기 기법은 공간지역성을 이용해 인접한 블록들을 함께 읽어옴으로써 저장장치와 파일시스템의 읽기성능을 향상시키는 기법이다. 그러나 비압축파일시스템과는 달리 압축파일시스템에서는 미리읽기 기법이 오히려 읽기 성능을 저하시키는 현상을 보인다. 그 이유는 압축파일시스템에서는 미리읽기의 미스패널티가 크게 증가하기 때문이다. 미스패널티는 미리 읽어온 블록이 미래에 실제로 사용되지 않는 경우에 발생하며 블록 하나를 읽어오는데 소모된 비용과 일치한다. 압축파일시스템에서는 블록을 저장장치에서 읽어오는 비용 이외에도 저

· 본 연구는 2009년 정부(교육과학기술부)의 재원으로 한국학술진흥재단의 지원을 받아 수행되었음(KRF-2009-0076335). 또한 이 연구를 위해 연구장비를 지원하고 공간을 제공한 서울대학교 컴퓨터연구소에 감사드립니다.
· 이 논문은 제36회 추계학술발표회에서 "커널의 미리읽기를 고려한 압축파일시스템의 읽기성능향상"의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 서울대학교 컴퓨터공학부
syahn@oslab.snu.ac.kr
†† 비 회 원 : 서울대학교 컴퓨터공학부
kakjagi@gmail.com
††† 종신회원 : 서울대학교 컴퓨터공학부 교수
kernkoh@oslab.snu.ac.kr

논문접수 : 2009년 12월 24일
심사완료 : 2010년 3월 28일

장장치에서 읽어온 압축블록을 압축해제하는 비용이 추가적으로 발생하게 된다. 특히 상대적으로 느린 CPU를 사용하는 모바일 장치에서는 압축해제에 더 많은 시간이 필요하며 미리읽기로 인한 압축파일시스템의 성능 저하 역시 더욱 심각해진다.

본 논문에서는 커널의 미리읽기를 고려해 압축파일시스템의 읽기성능을 향상시킬 수 있는 기법을 제안한다. 이 기법은 커널의 미리읽기를 그대로 활용하면서 압축파일시스템의 불필요한 압축해제비용을 제거해 압축파일시스템의 읽기성능을 향상시킨다. 우리의 실험결과에 따르면 본 논문에서 제안된 기법은 압축파일시스템의 읽기성능을 약 28%까지 향상시킬 수 있다.

본 논문은 다음과 같이 구성된다. 2장에서는 미리읽기 기법의 효과와 압축파일시스템의 성능 저하에 대하여 분석하고, 3장에서는 본 논문에서 제안하고 있는 기법에 대해서 소개한다. 4장에서는 3장에서 제안된 기법을 구현 및 실험을 통해 검증하고 마지막으로 5장에서 결론으로 마무리한다.

2. 압축파일시스템과 미리읽기 기법의 효과

2장에서는 미리읽기 기법의 효과와 그로 인해 압축파일시스템의 읽기성능이 저하되는 원인을 분석한다. 이를 위해 각각 대표적인 비압축파일시스템과 압축파일시스템인 EXT3[4]와 CramFS에서 메이저 폴트 처리시간을 측정하였다. 저장장치는 모바일 장치에서 널리 사용되는 낸드 플래시 메모리를 사용하였으며 Qtopia GUI[5] 부팅 시 발생하는 메이저 폴트들의 처리시간을 측정하였다.

그림 1과 그림 2는 각각 EXT3와 CramFS에서 최대 미리읽기 크기를 변화시키면서 측정한 총 메이저폴트 처리시간을 나타낸다. 그림 1을 보면 EXT3에서는 최대 미리읽기 크기가 증가할수록 메이저 폴트 처리 시간이 감소하는 것을 확인할 수 있다. 반면 CramFS에서는 미리읽기를 많이 하면 할수록 메이저 폴트 처리 시간은 오히려 증가하고 있다. 즉, 이 실험결과에서 우리는 미리읽기를 사용할 경우 비압축파일시스템에서는 읽기성능이 향상되지만 압축파일시스템에서는 오히려 읽기성능이 저하된다는 것을 알 수 있다.

미리읽기의 가장 큰 효과는 저장장치의 읽기성능 향상이다. 표 1과 그림 3을 보면 미리읽기를 사용한 경우 읽어온 페이지의 개수가 약 52% 증가하였지만 디바이스 드라이버 계층에서 소모되는 시간은 오히려 크게 줄어든 것을 확인할 수 있다. 이는 미리읽기 기법을 사용하면 한번에 읽어오는 블록의 양이 증가하여 저장장치의 읽기작업 처리량(read throughput)이 향상되기 때문이다. 즉, bulk read의 효과이다. 표 1을 보면 미리읽기를 사용하는 경우 저장장치에서 한번에 읽어오는 평균

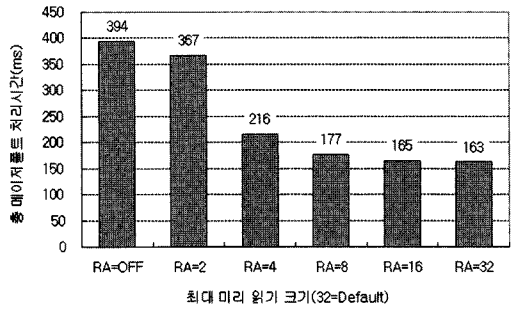

그림 1 EXT3에서 Qtopia GUI 부팅 시 총 메이저 폴트 처리 시간

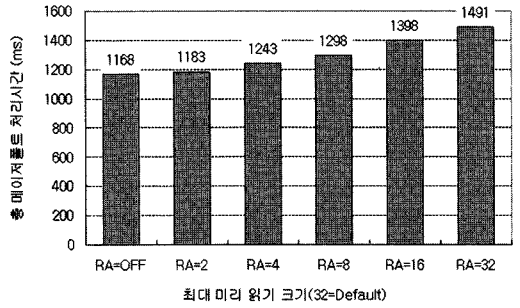

그림 2 CramFS에서 Qtopia GUI 부팅 시 총 메이저 폴트 처리 시간

표 1 EXT3와 CramFS에서 Qtopia GUI 부팅 시 각종 시스템 통계 자료

	EXT3		CramFS	
	RA=OFF	RA=32	RA=OFF	RA=32
메이저폴트 횟수	1245	108	1245	108
읽어온 페이지 개수	1245	1896	1245	1896
디바이스 작동횟수	1270	132	738	834
평균 읽기 크기(KB)	4	49	4	4

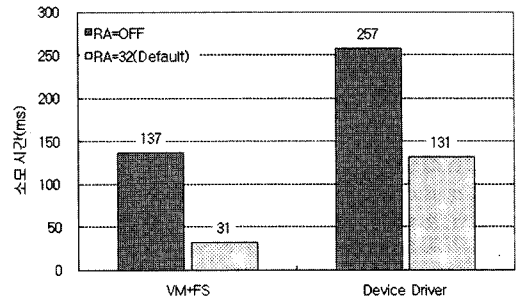

그림 3 EXT3에서 Qtopia GUI 부팅 시 각 계층에서 메이저 폴트 처리에 소모되는 시간

읽기크기가 증가하였다는 것을 확인할 수 있다.

반면 미리읽기가 압축파일시스템의 읽기성능을 저하

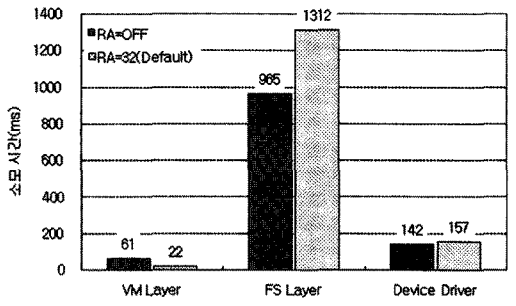

그림 4 CramFS에서 Qtopia GUI 부팅 시 각 계층에서 메이저 폴트 처리에 소모되는 시간

시키는 이유는 불필요한 압축해제로 인해 미스패널티가 너무 커지기 때문이다. 그림 4는 CramFS에서 메이저 폴트 처리시간을 각 계층별로 나타낸 그래프이다. 이 그래프에서 미리읽기를 사용하는 경우 파일시스템 계층에서 가장 큰 성능저하가 일어났음을 알 수 있다. 이 현상은 미리 읽어온 블록들 중에 실제 사용되지 않는 블록들이 불필요하게 압축해제 되었기 때문이다. 표 1을 보면 미리읽기를 사용한 경우 저장장치에서 읽어온 페이지들 중에 35%는 실제로 사용되지 않았다.

즉, 35%의 페이지들은 불필요하게 압축해제 되었으며 이 비용이 압축파일시스템의 미리읽기 미스패널티를 증가시키는 것이다.

미스패널티는 미리 읽어온 페이지가 실제로 사용되지 않을 경우 발생하며 해당 페이지를 저장장치로부터 읽어오는데 필요한 비용과 동일하다. 따라서 비압축파일시스템과 압축파일시스템의 미리읽기 미스패널티는 아래와 같다.

비압축파일시스템의 미스패널티 = 저장장치의 I/O 오버헤드

압축파일시스템의 미스패널티 = 저장장치의 I/O 오버헤드 + 압축해제비용

즉, 압축파일시스템에서 미리읽기 미스패널티는 장치 읽기비용 뿐만 아니라 압축해제비용까지 포함한다. 따라서 압축파일시스템에서 미리읽기 기법을 사용할 경우 미리읽기의 미스패널티가 미리읽기로 얻을 수 있는 저장장치의 성능향상을 상쇄시키는 것이다.

3. 커널의 미리읽기 기법을 고려한 압축파일시스템

3.1 디자인 목표

앞에서 살펴본 것과 같이 커널의 미리읽기 기법은 압축파일 시스템의 읽기성능을 저하시킨다. 이 문제를 해결하기 위해 다음과 같은 해결책을 고려할 수 있다.

첫 번째는 커널의 미리읽기를 사용하지 않는 방법이다. 가장 간단한 방법이지만 다른 파일시스템에도 영향을 끼치고 저장장치의 성능향상이라는 미리읽기의 이점을 이용할 수 없다는 한계를 가진다.

두 번째는 커널 미리읽기 기법의 적중률을 향상시키는 방법이다. 만약 적중률이 100%에 가까워진다면 미스패널티가 거의 발생하지 않게 되고 이에 따라 미리읽기로 인한 성능저하 문제도 해결된다. 미리읽기의 적중률을 높이기 위해 많은 연구들[6]이 진행되었지만 미리읽기의 적중률은 워크로드에 따라 달라지기 때문에 적중률을 높이는 데는 어느 정도 한계가 있다. 현재 리눅스 커널의 경우 미리읽기 기법의 적중률은 65% 정도에 불과하다.

마지막으로 압축파일시스템의 미리읽기 미스패널티의 크기를 줄이는 방법이다. 2장에서 살펴본 것과 같이 압축파일시스템의 읽기성능저하는 불필요한 압축해제로 인해 과도하게 증가한 미리읽기 미스패널티가 주요한 원인이다. 따라서 우리는 본 논문에서 압축파일시스템의 미리읽기 미스패널티의 크기를 줄일 수 있는 간단하고 효과적인 방법을 제안하고자 한다. 이를 위해 다음과 같은 디자인 목표를 제시한다.

첫 번째는 디바이스의 읽기성능을 향상시켜야 한다. 앞에서 미리읽기 기법이 저장장치의 읽기성능을 향상시키는 것을 확인하였다. 따라서 우리는 커널의 미리읽기 기법을 손상시키지 않고 활용할 수 있어야 한다.

두 번째는 압축해제 비용을 줄여야 한다. 커널의 미리읽기 기법에 의해 미리 읽어온 압축블록들은 지금 당장 필요한 데이터를 포함하고 있지 않으며 경우에 따라서는 긴 시간 동안 필요하지 않을 수도 있다. 따라서 우리는 미리 읽어온 압축블록들의 압축해제를 지연시킬 수 있다.

3.2 디자인 및 구현

앞에서 살펴본 디자인 목표들을 바탕으로 우리는 커널의 미리읽기를 고려한 압축파일시스템의 읽기 기법을 제안하고자 한다. 그림 5는 제안하고자 하는 기법을 간략하게 보여주고 있다. 제안된 기법은 다음 두 가지 단계로 이루어지며 커널의 미리읽기 기법을 손상시키지 않으면서도 미리읽기 기법으로 인한 불필요한 미스패널티를 줄여 압축파일시스템의 읽기 성능을 향상시킬 수 있다.

첫 번째 단계에서는 저장장치의 읽기 성능을 향상시키기 위해 저장장치에서 압축블록들을 한꺼번에 읽어온다. 이때 읽어오는 압축블록들은 커널의 미리읽기 기법에 의해 결정된다.

두 번째 단계에서는 저장장치에서 읽어온 압축블록들 중 지금 요청된 페이지를 포함하는 압축블록에 대해서만 압축해제를 수행한다. 미리읽기 페이지들은 현재 작업을 진행하는데 필요없으므로 미래에 실제로 사용자나 프로그램에 의해 요청되는 시점까지 압축해제를 지연시킨다.

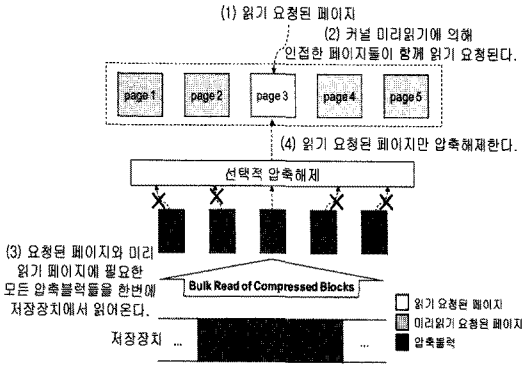

그림 5 커널의 미리읽기 기법을 고려한 압축파일시스템의 읽기 기법

이 선택적 압축해제 방법을 통해 우리는 미리읽기 페이지들의 미스패널티에서 압축해제 비용을 제거할 수 있다. 또한, 우리는 현재 읽기 요청된 페이지와 미리읽기 페이지들을 구분하기 위해 리눅스 페이지 구조체의 사용되지 않는 플래그를 활용하였다.

4. 실험 및 결과

4.1 실험 내용 및 환경

앞 장에서 제안한 기법의 성능을 검증하기 위해 우리는 리눅스 커널 2.6.21에서 CramFS를 수정하였으며 다음 3가지 버전의 압축파일시스템에 대하여 실험을 진행하였다.

CramFS - 리눅스 시스템에서 널리 사용되고 있는 압축파일시스템이며 커널의 미리읽기 기법을 그대로 사용한다.

CramFS(RAOFF) - 커널의 미리읽기 기법을 사용하지 않는 CramFS이다.

FCramFS - 3장에서 제안된 기법을 사용하도록 수정된 CramFS이다. 커널의 미리읽기 기법을 그대로 사용한다.

실험은 각 파일시스템에서 Qtopia GUI를 부팅하는 시간과 QPDF를 로딩하는 시간 동안 발생하는 메이저 폴트의 처리 시간을 측정하였다. 또한 각 계층별로 소모되는 시간을 측정하여 분석을 시도하였다. 성능 측정은 OMAP2420 플랫폼[7]에서 수행하였으며 저장장치는 SAMSUNG OneNAND 1G(KF1G16Q2M-DEB6)[8] 플래시 메모리를 사용하였다.

4.2 실험 결과

그림 6은 Qtopia GUI를 부팅하는 동안 발생하는 메이저 폴트의 처리 시간을 보여준다. 제안된 기법을 사용하는 FCramFS는 CramFS보다 28%까지 메이저 폴트 처리시간을 단축시킬 수 있었다. 또한 커널의 미리읽기

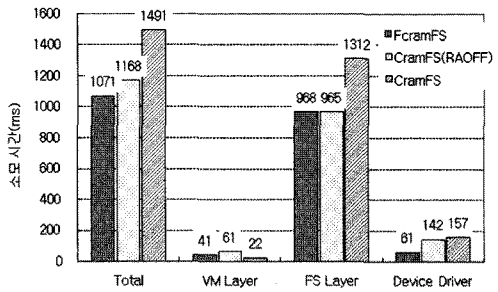

그림 6 Qtopia GUI 부팅 시 총 메이저 폴트 처리 시간과 각 계층별 소모시간

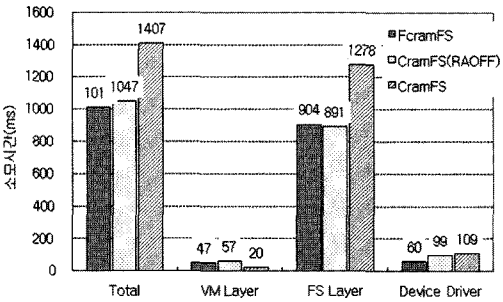

그림 7 QPDF 로딩 시 총 메이저 폴트 처리 시간과 각 계층별 소모시간

를 사용하지 않는 CramFS(RAOFF)에 비해서도 약 8%정도 메이저 폴트 처리 시간을 단축할 수 있었다. 이는 커널의 미리읽기 기법을 사용하는 것이 읽기성능 향상에 더 유리하다는 것을 보여준다. 그림 7은 QPDF를 로딩하는 시간 동안 발생하는 메이저 폴트의 처리시간을 측정한 그래프이다. 그림 6과 동일하게 FCramFS가 가장 빠른 읽기 성능을 보여주고 있다.

또한 그림 6과 7은 각 계층별로 소모되는 시간을 보여주고 있다. 그래프에서 볼 수 있듯이 FCramFS의 성능향상은 주로 파일시스템 계층에서 일어난다. 그림 6에 따르면 파일시스템 계층에서 소모되는 시간은 CramFS와 비교할 때 최대 25% 감소되었다. 이 값은 커널의 미리읽기 기법을 사용하지 않는 CramFS(RAOFF)의 파일시스템 계층 소모시간과 거의 동일한 측정값이며 이는 FCramFS가 커널 미리읽기 기법으로 인해 발생하는 추가적인 압축해제 오버헤드를 효과적으로 제거하고 있다는 것을 의미한다. 표 2를 보면 FCramFS에서 압축해제된 페이지의 개수는 CramFS의 67%에 불과하다. 그림 7의 계층별 소모시간 그래프 역시 동일한 실험 결과를 보여주고 있다.

FCramFS의 또 다른 성능향상은 디바이스 드라이버 계층에서 일어난다. 그림 6에 따르면 FCramFS은 CramFS

표 2 Qtopia GUI 부팅 시 각종 시스템 통계 자료

	FCramFS	CramFS(RAOFF)	CramFS
메이저 폴트 횟수	1244	1245	108
압축해제한 페이지 개수	1280	1245	1896
디바이스 작동횟수	183	738	834
평균 읽기 크기	14.5	4	4

에 비해 디바이스 드라이버 계층의 오버헤드를 61%까지 감소시켰다. FCramFS가 CramFS와 CramFS(RAOFF)보다 디바이스 드라이버 계층의 오버헤드를 줄일 수 있었던 이유는 한 번에 더 많은 압축블록들을 저장장치에서 읽어오기 때문이다. 표 2를 보면 FCramFS의 평균 읽기 크기는 14.5KB로 CramFS와 CramFS(RAOFF)보다 약 3.5배 더 크다. 이는 제안된 기법이 bulk read를 통해 저장 장치의 읽기성능을 향상시켰다는 것을 의미한다. 동일한 실험 결과를 그림 7에서도 확인할 수 있다.

반면 표 2를 보면 FCramFS의 경우 CramFS에 비해 메이저 폴트 횟수가 크게 증가하였다. FCramFS에서는 압축블록들을 미리 읽어오지만 미리읽기 페이지들은 압축해제를 하지 않는다. 따라서 가상메모리 계층에서는 미리읽기 페이지들을 메모리에서 찾을 수 없으며 미리읽기로 인한 메이저 폴트 횟수 감소효과 역시 얻을 수 없다. 이로 인해 메이저폴트 횟수가 증가하고 그림 6과 7에서처럼 가상메모리 계층의 오버헤드가 소폭 상승하였지만 전체 오버헤드에 비해 미미한 수준이므로 불필요한 압축해제를 제거해 파일시스템 계층의 오버헤드를 줄이는 것이 성능상 훨씬 유리하다.

앞에서 살펴본 실험 결과들을 바탕으로 우리는 제안된 기법이 bulk read와 선택적 압축해제를 통해 압축파일시스템의 읽기 성능을 향상시킬 수 있다는 사실을 확인할 수 있다.

5. 결 론

본 논문에서 우리는 커널의 미리읽기 기법이 압축파일시스템의 읽기성능을 저하시킨다는 점을 지적하고 문제점을 개선하기 위해 커널의 미리읽기 기법을 고려한 압축파일시스템의 읽기 기법을 제안하였다. 제안된 기법은 bulk read와 선택적 압축해제를 통해 커널의 미리읽기 기법을 손상시키지 않으면서 미리읽기 미스패널티를 줄이는 기법이다. 실험 결과 제안된 기법은 기존의 압축파일시스템에 비해 메이저 페이지 폴트 처리시간을 28%까지 감소시킬 수 있었다.

참 고 문 헌

[1] W. T. Huang, C. T. Chen, Y. S. Chen, and C. H. Chen, "A compression layer for NAND type flash memory systems," *Proc. of the IEEE ICITA 2005*, vol.02, pp.599-604, 2005. (in Australia)

[2] CramFS document, http://lxr.linux.no/source/fs/cramfs/README

[3] CramFS tools, http://sourceforge.net/projects/cramfs/

[4] S. Tweedie, "EXT3, Journaling Filesystem," *Proc. of the 2000 Ottawa Linux Symposium*, 2000. (in Canada)

[5] Qtopia Platform, http://doc.trolltech.com/qtopia2.2/html/overview.html

[6] M. Opdenacker, "Readahead: time-travel techniques for desktop and embedded systems," *Proc. of the 2007 Ottawa Linux Symposium*, vol.02, pp.97-106, 2007. (in Canada)

[7] TI OMAP 2420 Platform, http://focus.ti.com/general/docs/wtbu/wtbuproductcontent.tsp?templateId=6123&navigationId=11990&contentId=4671

[8] Samsung OneNANDTM KFXG16Q2M-DEB6 datasheets, http://www.samsung.com/global/business/semiconductor/productInfo.do?fmly_id=160&partnum=KFG1G16Q2M#component03