동적 XIP(eXecute In Place)를 위한 비용 인식 캐시 알고리즘 설계

김도훈, 박찬익
포항공과대학교 컴퓨터공학과
e-mail:{hunkim,cipark}@postech.ac.kr

Cost-Aware Cache Algorithm for Dynamic XIP (eXecute In Place)

Dohun Kim, Chanik Park
Department of Computer Science and Engineering, POSTECH

요 약

본 논문은 기존의 XIP 기법에서 발생할 수 있는 메모리 접근 성능저하를 해결하기 위한 동적 XIP 기법을 제안하였다. 동적 XIP 기법은 상대적으로 성능저하가 적을 것으로 예상되는 코드 페이지들을 동적으로 선택하여 XIP 영역으로 설정하고, 성능저하가 크게 나타날 것으로 예상되는 코드 페이지들을 램 캐시에 캐싱하여 성능을 향상시킨다. 본 논문은 램 캐시를 관리하기 위해 MIN 캐시 알고리즘 및 메모리 접근 비용을 고려한 오프라인 캐시 알고리즘과, 페이지 접근에 대한 최신성(Recency) 및 슬라이딩 윈도우에 저장된 페이지 접근 기록에 기반하여 메모리 접근 비용을 예측하는 온라인 캐시 알고리즘, 온라인 캐시 알고리즘의 램 캐싱 판단의 정확성을 높이는 기법을 제안하였다. 본 논문은 온・오프라인 알고리즘의 성능비교를 위해 시뮬레이터를 통해 성능을 평가하였고, 유용성을 시험하기 위해 온라인 알고리즘을 리눅스를 기반으로 구현하여 성능을 평가하였다. 본 논문에서 제안한 동적 XIP는 실제 구현한 환경에서 실험한 결과, 작은 크기의 캐시를 사용하고도 수행시간에서는 최대 27%, 에너지 소모량에서는 최대 24%의 성능이 향상됨을 보였다.

1. 서론

모바일용 내장형 시스템은 XIP를 통해 플래시 메모리를 저장장치로 사용할 뿐만 아니라, 메모리 사용량, 전력 소모량, 소프트웨어 적재 시간 등을 감소시킬 수 있다[1]. 그러나 플래시 메모리의 하드웨어적인 특성에 기인한 XIP의 메모리 접근 성능저하 문제는 연구자들에게 이미 알려져 있는 사실이나, 이를 해결하기 위한 구체적인 방법은 제시되고 있지 않고 있다[2].

본 논문은 플래시 메모리에 대한 XIP의 메모리 접근 성능저하 문제를 소프트웨어 적인 방법으로 향상시킬 수 있는 동적 XIP 기법을 제안하였다. 동적 XIP 기법은 코드 페이지의 접근 패턴에 따라 코드 페이지의 접근을 램 혹은 플래시 메모리에서 발생하도록 설정한다. 특히 메모리 접근 성능의 저하가 크게 나타날 것으로 예상되는 코드 페이지들은 정해진 크기의 램 캐시에 캐싱하여 XIP의 성능을 향상시킨다. 따라서 메모리 접근이 램과 플래시 메모리에서 모두 발생한다.

본 논문은 제한된 크기의 램 캐시를 활용하기 위하여 온・오프라인 캐시 알고리즘을 제안하였다. 그리고 오프라인과 온라인 알고리즘의 성능을 비교하기 위해 시뮬레이터를 사용하여 성능을 평가하였다. 그리고 제안된 온라인 알고리즘의 유용성을 시험하기 위해 리눅스를 기반으로 하여 임베디드 플랫폼에 구현하였다. 실험 결과 본 논문에서 제안된 동적 XIP 기법은 구현된 환경에서 응용의 전체 코드 페이지 수에 비해 작은 캐시를 사용하고도 기존 XIP에 비해 수행시간에서 최대 27%, 에너지 소모량에서 최대 24%의 성능이 향상됨을 보였다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구에 대해 설명한다. 3장은 본 논문에서 제안한 캐시 알고리즘을 설명한다. 4장은 제안된 알고리즘의 실험 결과를 설명한다. 마지막으로 5장에서는 결론을 맺는다.

2. 관련 연구

XIP 기법은 노어(NOR) 플래시 메모리에서 주로 사용되고 있다. 그러나 최근 가격대 성능비가 우수한 낸드(NAND) 플래시 메모리에서도 XIP를 지원하기 위한 연구가 시도되었다[3]. 원낸드(OneNAND) 플래시 메모리에서도 동적 XIP를 지원하기 위한 연구가 시도되었다[4]. 이 연구는 요구 페이징시 발생하는 페이지 복사 오버헤드를 줄이기 위해 동적으로 XIP 기능을 적용하였다. 그러나 OneNAND라는 특수한 하드웨어 환경을 가정하고 있을 뿐만 아니라, 페이지 복사 오버헤드만을 고려하고 있고, 이를 줄이기 위한 구체적인 기준을 제시하지 않았다.

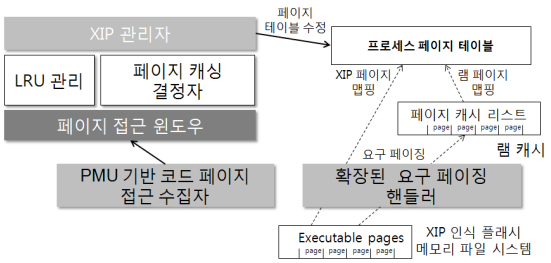
※ 본 연구는 대학 IT 연구센터 육성지원사업의 연구결과로써 HY-SDR 연구센터의 연구비 지원으로 수행되었음.

비용 인식 캐시 알고리즘은 페이지 캐싱에 대한 판단이 틀렸을 때 발생하는 성능상 불이익을 기준으로 캐싱을 결정하는 알고리즘이다. 기존 캐시 알고리즘[5, 6]은 모든 메모리 접근이 램에서만 발생하는 것으로 가정하고 있어 동적 XIP 환경에 적용할 수 없다.

본 논문에 제안하는 동적 XIP는 다음과 같은 특징을 가지고 있다. 1) 동적 XIP를 위한 비용 인식 캐시 알고리즘을 제안하였다. 2) 동적 XIP 기법을 실제 시스템에 구현하여 유용성을 보였다. 3) 특정 하드웨어가 아닌 일반적인 임베디드 플랫폼 환경에서도 사용할 수 있다.

3. 동적 XIP를 위한 캐시 알고리즘

3.1 동적 XIP 구조



(그림 1) 온라인 캐시 알고리즘을 위한 동적 XIP 구조도

본 논문은 XIP 적용시 발생하는 응용의 메모리 접근 성능저하를 개선하기 위해 동적 XIP를 제안하였다. 일반적으로 접근 빈도가 높은 페이지일수록 메모리 접근 성능에 큰 영향을 준다. 따라서 동적 XIP는 CPU에서 제공하는 PMU(Performance Monitoring Unit)를 사용하여 메모리 접근 정보를 수집하는 코드 페이지 접근 수집자를 포함하고 있다. 수집된 페이지 접근 로그는 윈도우 형태로 관리된다. 램에 캐싱된 페이지에 대한 최신성 정보를 수집하기 위해 LRU로 페이지들을 관리한다. 페이지 캐싱 결정자는 페이지의 최신성과 윈도우에 저장된 메모리 접근 로그에 따라 향후 메모리 접근 성능을 향상 시킬 수 있는 XIP 영역의 페이지와 상대적으로 성능 향상도가 떨어지는 램 캐시의 페이지를 선택한다. 마지막으로 XIP 관리자는 결정된 페이지 교체를 적용하기 위해 해당 프로세스의 페이지 테이블을 수정한다. 확장된 요구 페이징 핸들러는 수정된 페이지에 대한 접근 발생시 각 페이지에 대한 페이지 테이블 맵핑을 수정한다.

3.2 오프라인 비용 인식 캐시 알고리즘(CA-MIN-dxip)

동적 XIP를 위한 오프라인 비용 인식 캐시 알고리즘은 미래의 접근 정보가 알려져 있을 때 가장 높은 캐시 적중률을 보이는 MIN 알고리즘[7]에 기반하고 있다.

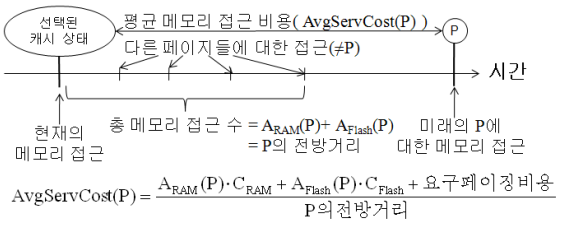
3.2.1 평균 메모리 접근 비용

동적 XIP를 위한 비용 인식 캐시 알고리즘에서 비용이란 평균 메모리 접근 비용을 의미한다(그림 2). 예를 들어 페이지 P를 페이지 교체의 대상으로 선택하면, 현재의 시점에서 선택된 캐시 상태는 페이지 P에 대한 접근이 발생할 때까지 유지될 것이다. 따라서 페이지 P의 다음 접근이 발생할 때까지 예상되는 총 페이지 접근 수는 A_{RAM}(램에서 발생하는 총 페이지 접근 수)과 A_{Flash}(플래시 메모리에서 발생하는 총 페이지 접근 수)만큼 발생한다. 오프라인 알고리즘에서 페이지 P의 접근 시점은 MIN 알고리즘의 전방거리(Forward Distance)로 표현할 수 있다. 그리고 P에 대한 다음 접근이 발생할 때까지 발생한 메모리 접근 비용은 다음과 같이 계산할 수 있다.

$$A_{RAM}(P) \cdot C_{RAM} + A_{Flash}(P) \cdot C_{Flash} + \text{요구페이징비용}$$

이때 C_{RAM}과 C_{Flash}는 캐시 알고리즘에서 간주하는 비용이 시간이라면 메모리 접근당 소요시간을, 에너지 소모량이라면 메모리 접근당 소요 에너지량을 의미한다. 요구페이징 비용은 P를 교체하는데 소모된 시간 혹은 에너지 소모량을 의미한다.

따라서 현재 캐시 상태를 선택했을 때 시스템에서 향후 발생할 메모리 접근 당 메모리 접근 비용을 의미한다.



(그림 2) 동적 XIP의 평균 메모리 접근 비용 계산 구간

```
Input : page access r and current cache content C
if r ∈ C  // cache hit
    return;

// make a state set of new cache contents
𝒞 ← C + {r}
make all possible cache state set 𝒞 from 𝒞 that the size equals to |C|

for each cache state C ∈ 𝒞 do
    p ← 𝒞 − C  // p means a victim page
    // estimate average service cost for each victim p
    calculate AvgServCost(p) = service_cost(p) / forward_distance(p)

find a cache state C ∈ 𝒞 with minimum AvgServCost(p)
if r == p return;
choose p as a victim page and update current cache content( C ← C )
```

(그림 3) 오프라인 비용 인식 캐시 알고리즘

3.2.2 비용 인식 캐시 알고리즘

비용 인식 캐시 알고리즘의 동작은 다음과 같다. 현재 시점에서 생성할 수 있는 캐시 상태를 결정한다. 이때 생성할 수 있는 캐시 상태에는 페이지 교체가 일어나지 않은 경우도 포함한다. 즉, 현재 접근된 페이지가 XIP 영역으로 접근될 페이지로 결정된 것을 의미한다. 이제 XIP

영역으로 결정된(교체된) 페이지의 다음 접근시점까지 평균 메모리 접근 비용을 계산한다. 그리고 가장 작은 평균 메모리 접근 비용이 예상되는 캐시 상태를 선택하여 해당 페이지를 교체한다. 동적 XIP는 램이 아닌 플래시 메모리에서도 페이지 접근이 가능하므로, 페이지 교체 후 메모리 접근의 성능 향상이 기대되지 않을 경우 현재 상태를 그대로 유지한다. 그림 3은 오프라인 비용 인식 캐시 알고리즘에 대해 기술하였다.

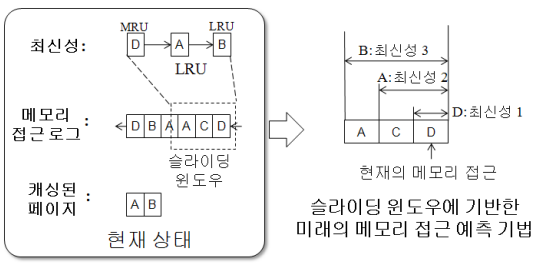
3.3 온라인 비용 인식 캐시 알고리즘(OCA-MIN-dxip)

3.3.1 오프라인 알고리즘에 대한 근사기법

온라인 알고리즘은 미래에 대한 메모리 접근 정보가 존재하지 않으므로, 평균 메모리 접근 비용 계산에 필요한 페이지의 전방거리와 전방거리까지의 메모리 접근 수를 알 수 없다. 따라서 이를 해결하기 위한 근사기법이 필요하다.

온라인 알고리즘에서 전방거리를 예측하기 위해 가장 많이 사용되는 방법으로 LRU 기반 페이지 관리법을 들 수 있다. 본 논문에서도 전방거리 예측을 위해 LRU를 사용한다. LRU로 관리되는 페이지에는 램 캐시에 있는 페이지 뿐만 아니라 현재 접근된 페이지(플래시 메모리에 있는)도 포함한다. 그리고 MRU 페이지부터 각 페이지에 최신성 값을 1부터 순차적으로 할당한다.

다음으로 본 논문은 전방거리까지 발생하는 메모리 접근 수를 예측하기 위해 슬라이딩 윈도우를 기반으로 하는 페이지 접근 로그를 사용한다. 윈도우 크기가 작을 경우, 가장 최근의 접근 수만이 반영되고, 윈도우 크기가 클 경우, 먼 과거의 접근 수까지 반영이 될 수 있다. 그러나 적당한 윈도우 크기를 정하는 문제는 온라인 알고리즘에서 다루기 어려운 부분이다. 따라서 본 논문은 LRU에서 관리하는 페이지 수를 윈도우 크기로 정하였다.



(그림 4) 윈도우 기반 미래 메모리 접근 수 예측 기법

그림 4는 LRU에서 수집된 각 페이지의 최신성을 기반으로 미래의 페이지 접근 수를 예측하는 기법을 예를 통해 도시한 것이다. 현재 접근된 페이지 D는 LRU에서 최신성이 1인 위치에 있으며, 현재 시점을 기준으로 과거의 1개 메모리 접근(D)를 미래의 메모리 접근 수로 예측한다. 페이지 B는 최신성이 3인 위치에 있으며, 현재 시점을 기준으로 과거 3개 메모리 접근(A, C, D)을 미래의 메모리 접근 수로 예측한다.

다음으로 현재 시점에서 생성할 수 있는 캐시 상태들을 찾아 낸 후 각 상태에 해당되는 평균 메모리 접근 비용을 다음과 같은 식을 통해 계산한다.

$$AvgServCost(p) = \frac{\text{예측된 접근 수에 따른 메모리 접근 비용}}{\text{페이지 } p\text{의 최신성}}$$

예를 들어 그림 4에서 생성할 수 있는 캐시 상태로 (A, D)가 있을 수 있다. 즉, 페이지 B가 교체대상이 되었을 때, 예상되는 평균 메모리 접근 비용은 다음과 같다.

$$AvgServCost(B) = \frac{2 \cdot C_{RAM} + 1 \cdot C_{Flash} + \text{요구페이징비용}}{3}$$

3.3.2 페이지 캐싱에 대한 정확도 향상 기법

온라인 비용 인식 캐시 알고리즘은 최신성과 윈도우 기반 메모리 수 예측으로 평균 메모리 접근 비용을 계산한다. 따라서 램 캐싱 판단에 대한 정확성이 부족하다. 본 논문은 다음과 같은 검정법을 통해 정확성을 향상시킨다.

온라인 알고리즘에 의해 교체될 페이지를 j, 램 캐싱될 페이지를 i라 하자. 각 페이지에 대한 미래의 접근 수가 알려져 있을 때, 이를 각각 A_j, A_i라 한다. 그러면 메모리 접근 성능에 영향을 미치는 페이지는 당연히 두 페이지 밖에 없으므로 이 둘이 발생시키는 메모리 접근 성능을 살펴 본다. 페이지 교체를 유보할 경우, 앞으로 예상되는 메모리 접근 성능(C^{norep})은 다음과 같다.

$$C^{norep} = A_j \cdot C_{RAM} + A_i \cdot C_{Flash}$$

페이지를 교체할 경우, 앞으로 예상되는 메모리 접근 성능(C^{rep})은 다음과 같다.

$$C^{norep} = A_j \cdot C_{Flash} + A_i \cdot C_{RAM} + \text{요구페이징비용}$$

따라서 페이지 교체시 메모리 접근 성능이 향상되려면 다음의 조건식을 만족해야 한다.

$$C^{norep} \geq C^{rep}$$

$$A_i - A_j \geq \frac{\text{요구페이징비용}}{C_{Flash} - C_{RAM}} = A_{threshold}$$

일반적으로 응용이 플래시 메모리와 램을 접근하는 비용과 요구페이징 비용은 계산을 통해 구할 수 있다. 따라서 조건식은 램 캐싱될 페이지가 교체될 페이지에 비해 일정 수 이상의 메모리 접근 수가 많으면 램 캐싱을 하는 것이 메모리 접근 성능을 향상시킬 수 있음을 의미한다. 따라서 온라인 알고리즘은 교체결정 마지막 단계에서 제시된 조건식을 통해 램 캐싱을 결정한다.

4. 실험

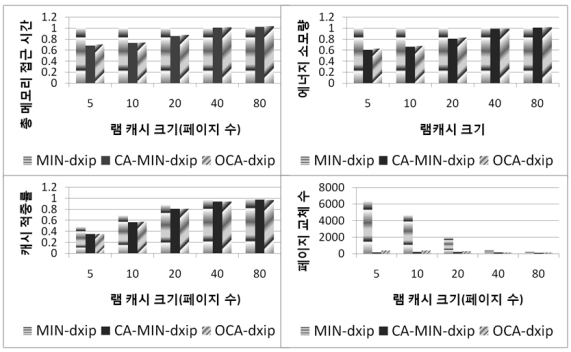
4.1 실험 환경

본 논문은 실험을 위해 작업부하 응용으로 미시간 대학에서 개발한 MiBench[8]를 사용하였다. 지면관계상 실험 결과 중 lout 응용의 결과만을 본 논문에 보였다. lout 응용은 총 225개의 코드 페이지로 구성되며, 실험 환경에서

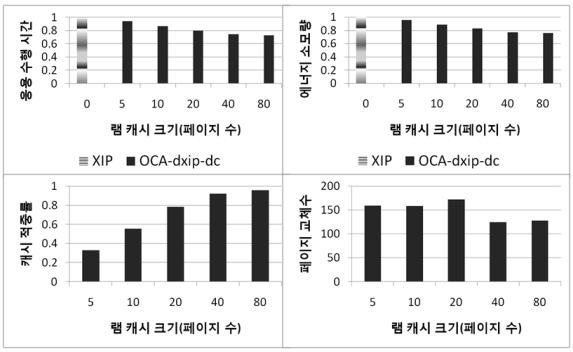
는 캐시 크기를 5~80개의 페이지 수로 변화시키며 성능을 측정하였다.

본 논문은 온·오프라인 캐시 알고리즘의 성능을 비교하기 위해 시뮬레이션을 사용하였다. 시뮬레이션에 사용된 lout 응용의 메모리 접근은 실제 하드웨어 플랫폼에서 측정된 것을 사용하였다. 그리고 온라인 알고리즘은 PXA255 CPU 및 64MB 노어 플래시 메모리가 탑재된 하드웨어와 리눅스 버전 2.6.10을 기반으로 구현하였고, 에너지 소모량은 디지털 멀티미터를 통해 초당 10000개의 샘플링을 통해 측정하였다.

4.2 실험 결과



(그림 5) 시뮬레이션 결과



(그림 6) 실제 환경의 실험 결과

그림 5는 시뮬레이션을 통해 온·오프라인 캐시 알고리즘의 성능을 비교한 것이다. 결과 중 MIN-dxip는 비용을 고려하지 않고 캐시 적중률만을 높인 오프라인 캐시 알고리즘이다. 본 논문에서 제안된 비용 인식 알고리즘은 비용을 고려하므로 MIN-dxip과는 달리 캐시 적중률을 희생하는 대신 페이지 교체수를 줄여 메모리 접근 시간이나 에너지 소모량에서 메모리 접근 성능을 향상시킨다.

그림 6은 실제 환경에서 온라인 캐시 알고리즘의 성능을 XIP와 비교한 것이다. XIP는 캐시를 사용하지 않으므로 캐시 적중률과 페이지 교체수의 결과는 표시하지 않았다. 제안된 온라인 알고리즘을 실험한 결과 캐시 적중률과 페이지 교체 수는 시뮬레이션 결과와 거의 동일한 결론을 얻었다. 그러나 실제 환경에서는 메모리 접근 시간과 메모리만의 에너지 소모량을 측정할 수 없으므로, 이를 응용 수행시간과 응용의 에너지 소모량으로 대체하였다. 실험 결과, 제안된 알고리즘은 작은 크기의 캐시를 사용하고도 수행시간은 최대 27%, 에너지 소모량은 최대 24%가 향상되었다.

5. 결론

본 논문은 XIP 사용시 발생할 수 있는 메모리 접근 성능저하를 향상시키는 동적 XIP 기법을 제안하였다. 동적 XIP는 제한된 크기의 램 캐시를 사용하므로 이를 효율적으로 관리하기 위한 온·오프라인 알고리즘을 제안하였다. 제안된 기법은 시뮬레이션과 실제 환경에서 구현하여 성능을 평가하였다. 실험 결과, 동적 XIP는 실제 환경에서 작은 크기의 캐시를 사용하고도 수행시간 면에서 최대 27%, 에너지 소모량면에서 최대 24%의 메모리 접근 성능을 향상시켰다.

참고문헌

[1] Jared Hulbert and Justin Treon, "Creating optimized XIP systems", CELF Embedded Linux Conference Presentation, 2006.

[2] Vitaly Wool, "XIP: the past, the present... the future?" Free and Open source Software Developers' European Meeting 2007

[3] Chanik Park and et al, "A Low-cost Memory Architecture with NAND XIP for Mobile Embedded Systems", Proceedings of the International Conference on Hardware-So ftware Codesign and System Synthesis, 2003.

[4] Yongsoo Joo, Yongseok Choi, Chanik Park, Sung Woo Chung, Eui-Young Chung and Naehy k Chang, "Demand Paging for OneNAND Flash eXecute-In-Place", Proceedings of the Internat ional Conference on Hardware-Software Codesi gn and System Synthesis, 2006.

[5] Young N.E, "On-line file caching", Proceedings of the 9th Annual ACM-SIAM symposium on Discrete Algorithms, (Balitmore), Jan. 17-19, 1999

[6] S. Liang, K. Chen, S. Jiang and X. Zhang, "Cost-Aware Caching Algorithms for Distributed Storage Servers", Proceeding of the 21st International Symposium on Distributed Computing, Sep. 24-26, 2007

[7] Belady. L, "A study of replacement algorithms for virtual storage computers", IBM System Journal vol. 5, no. 2, pp. 78-101, 1966

[8] Ringenberg and et al, "MiBench: A free, commerc -ially representative embedded benchmark suite", IEEE 4th Annual Workshop on Workload Characterization, Dec. 2001