The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 18, No. 5, pp.155-160, Oct. 31, 2018. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2018.18.5.155
JIIBC 2018-5-20

렌더링 시스템을 위한 버퍼캐쉬 관리기법

Management Technique of Buffer Cache for Rendering Systems

신동희*, 반효경**

Donghee Shin*, Hyokyung Bahn**

요 약 본 논문에서는 범용 시스템에서 널리 사용되는 버퍼캐쉬가 렌더링 소프트웨어 수행시 비효율적으로 동작한다는 것을 분석하고 이를 해결하는 렌더링 시스템용 버퍼캐쉬 관리 기법을 제안한다. 이를 위해 본 논문은 먼저 다양한 렌더링 파일 입출력 트레이스를 추출하여 렌더링의 고유한 입출력 패턴을 분석하였다. 그 결과 렌더링 파일 입출력은 주기가 긴 반복참조와 주기가 짧은 반복참조, 임의참조, 그리고 1회성 쓰기참조로 구성된다는 것을 발견하였다. 본 논문은 이러한 렌더링의 파일참조 특성을 고려하여 버퍼캐쉬 공간을 참조별로 할당하고 그에 맞는 관리 기법을 제안하여 기존 버퍼캐쉬 대비 평균 19%, 최대 55%의 성능을 개선시켰다.

키워드 : 렌더링, 버퍼캐쉬, 파일 입출력, 반복참조, 임의참조

Abstract In this paper, we found that buffer cache in general systems does not perform well in rendering software, and presented a new buffer cache management scheme that resolves this problem. To do so, we collected various file I/O traces of rending software and analyzed their characteristics. From this analysis, we observed that file I/Os in rendering consist of long loops, short loops, random accesses, and write-once accesses. Based on this observation, we presented a buffer cache management scheme that allocates cache space to each access types and manages them appropriately, thereby improving the buffer cache performances by 19% on average and up to 55%.

Key Words : Rendering, buffer cache, file I/O, loop, random access

Ⅰ. 서 론

렌더링이란 형태, 명암, 질감 등의 기본 입력파일을 바탕으로 초당 24 프레임 이상의 고화질 이미지를 소프트웨어 작업을 통해 생성하는 과정으로 3D 에니메이션, 특수효과 등에 널리 사용된다[1,2]. 렌더링에서는 하나의 프레임 생성에 길게는 이틀 이상의 시간이 소요될 정도로 많은 연산량을 필요로 하므로 PC가 아닌 전용 클러스터 머신을 사용하는 것이 일반적이다. 렌더링 작업이 연산 집약적인 특성을 나타내지만, 최근 매니코어 시스템을 통한 병렬 작업의 극대화로 CPU 또는 GPU에 비해 수십 만배 이상 느린 스토리지 접근이 렌더링 시스템의 또다른 병목점으로 인식되고 있다. 본 논문은 렌더링 시스템의 스토리지 성능 개선을 위한 효율적인 버퍼캐쉬 관리 기법을 제안한다. 버퍼캐쉬는 스토리지에서 읽어 들인 파일을 메모리에 저장하여 재요청시 빠른 서비스를 가능하게 한다[3]. 본 논문에서는 일반적인 컴퓨터 시스템에 탑재된 버퍼캐쉬가 렌더링 소프트웨어를 실행시킬 경우

*정회원, 이화여자대학교 컴퓨터공학과
**정회원, 이화여자대학교 컴퓨터공학과(교신저자)
접수일자 : 2018년 7월 26일, 수정완료 : 2018년 9월 6일
게재확정일자 : 2018년 10월 5일

Received: 26 July, 2018 / Revised: 6 September, 2018 / Accepted: 5 October, 2018
**Corresponding Author: bahn@ewha.ac.kr
Dept. of Computer Engineering, Ewha University, Korea

비효율적으로 동작한다는 것을 분석하고 이를 해결하는 새로운 관리 기법을 제안한다.

본 논문은 먼저 다양한 렌더링 파일 입출력 트레이스를 추출하고 이를 분석하였다. 그 결과 렌더링의 특성에 기반한 3가지 고유한 입출력 패턴을 발견하였다. 그 첫째는 렌더링 소프트웨어의 시작 시에 나타나는 환경설정파일 읽기로 이는 매 프로그램 실행 시 1번의 참조로 나타나지만 프로그램의 재실행시 재참조가 이루어져 주기가 긴 반복참조 형태를 나타내었다. 두번째는 렌더링 작업에 필요한 텍스쳐 입력파일 읽기로 이는 주기가 짧은 반복참조와 임의참조가 혼합된 형태로 나타났다. 세번째는 렌더링 결과 생성된 출력파일에 대한 쓰기로 이는 1회성 순차참조 형태로 나타났다.

본 논문은 이러한 렌더링의 파일참조 특성이 버퍼캐쉬의 성능을 크게 저하시킨다는 점을 분석하고 이를 해소하는 기법을 제안하였다. 다양한 렌더링 트레이스를 이용한 실험을 통해 제안하는 버퍼캐쉬 관리기법이 기존 시스템에 비해 평균 19%, 최대 55% 성능 개선 효과가 있는 것을 확인하였다.

II. 연구동기 및 관련연구

1. 연구 동기

그림 1과 2는 각각 범용 워크로드와 렌더링 워크로드를 LRU(least recently used) 알고리즘을 사용하는 일반적인 버퍼캐쉬 시스템에서 실행했을 때 캐쉬 크기에 따른 성능을 보여주고 있다. 그림에서 보는 것처럼 범용 워크로드는 캐쉬 크기가 증가할수록 성능이 점진적으로 개선되나, 렌더링 워크로드의 경우 버퍼캐쉬의 크기가 증가하더라도 성능이 거의 개선되지 않다가 데이터를 모두 담을 수 있을 정도로 캐쉬 크기가 증가한 후에야 성능 개선이 나타나는 것을 확인할 수 있다. 이는 렌더링 시스템의 버퍼캐쉬 관리시 범용 컴퓨터의 방식을 그대로 사용하는 것이 적절하지 않음을 뜻한다. 본 논문은 렌더링 워크로드에 적합한 버퍼캐쉬 관리 기법을 설계하여 캐쉬 크기 증가 시 이에 비례한 성능 개선이 나타날 수 있도록 하는 것을 목표로 한다.

2. 관련 연구

버퍼캐쉬의 성능 개선을 위한 다양한 연구가 있어 왔다. UBM은 참조패턴을 실시간으로 탐지하고 패턴별 성능 개선 효과에 기반해 캐쉬 공간을 할당하는 기법을 제안하였다[4]. 그러나, UBM은 다양한 혼합형 워크로드를 위한 버퍼캐쉬 관리 기법으로 참조패턴을 실시간으로 탐지하고 공간을 할당하기 위해 메모리상에 탐지 테이블을 유지하고 참조 정보를 갱신하는 오버헤드가 발생하는 제약이 있다.

ARC는 히스토리 버퍼를 두고 워크로드의 특성이 최근성과 접근 빈도 중 어느 쪽에 더 영향을 받는지를 조사하고 이를 바탕으로 캐쉬 공간을 조절한다[5]. 이 방법은 반복참조를 고려하지 않는다는 약점이 존재한다.

CRAW는 가상메모리 페이지 참조 중 읽기 연산과 쓰기 연산의 특성을 별도로 분석하고 이에 기반해서 연산별로 메모리 공간을 할당하는 방법을 사용한다[6]. 플래시

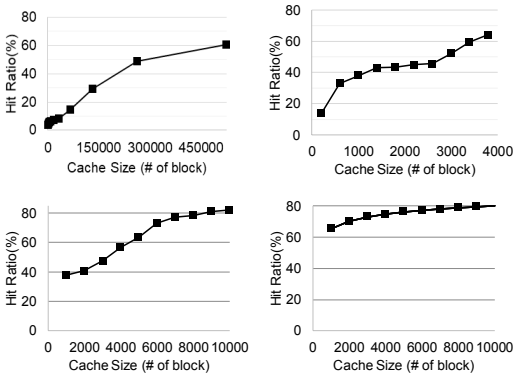


그림 1. 범용 시스템의 버퍼캐쉬 성능
Fig. 1. Buffer cache performance in general purpose systems.

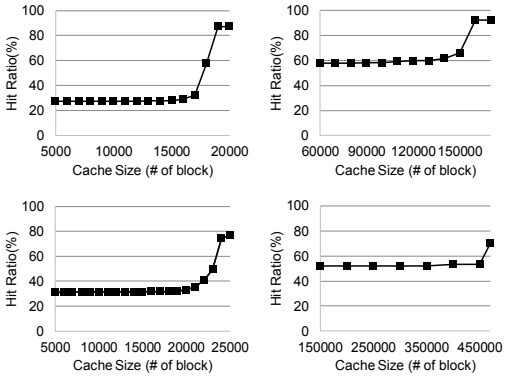


그림 2. 렌더링 시스템의 버퍼캐쉬 성능
Fig. 2. Buffer cache performance in rendering systems.

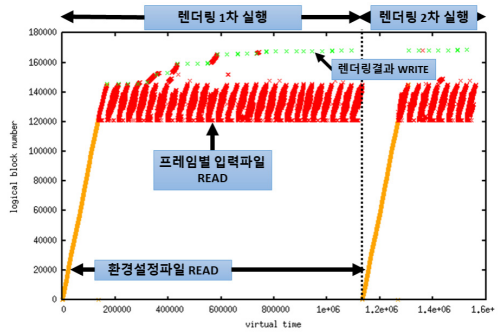
메모리라는 저장매체의 특성을 고려하나 워크로드 특성을 고려하지 않는다는 약점이 있다. 이처럼 기존 기법들은 대부분 혼합형 워크로드를 가정하거나 실시간으로 워크로드의 특성을 탐지해야 하는 오버헤드가 뒤따른다는 것이 본 연구의 방향과 차이점이다.

Ⅲ. 렌더링 시스템을 위한 버퍼캐쉬

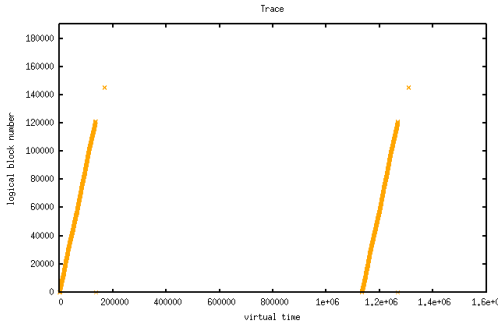
렌더링 시스템을 위한 버퍼캐쉬 설계를 위해 본 논문에서는 먼저 렌더링 실행시의 파일블록 참조 트레이스를 추출하고 분석하였다. 그림 3(a)는 추출한 트레이스에서 시간에 따라 참조되는 블록을 파일 유형별로 표시한 것이다. 그림에서 x축은 블록이 참조될 때마다 1씩 증가하는 논리적 시간이며, y축은 블록번호를 나타낸다. 블록 참조를 파일 유형별로 분석해보면 그림 3(b)의 환경설정파일 읽기가 프로그램 런치 시에 나타나며, 그림 3(c)의 텍스처 입력파일 읽기가 매 프레임마다 나타난다. 또한, 그림 3(d)의 렌더링 결과파일 쓰기가 매 프레임 렌더링이 끝날 때마다 발생한다. 이러한 특성은 어떠한 렌더링 작업을 실행하더라도 공통적으로 나타나는 것으로 확인되었다.

그림 3(b)의 환경설정파일 읽기는 매 프로그램 실행시마다 1번씩 나타나므로 주기가 긴 반복 참조형태를 띄며, 그림 3(c)의 텍스처 입력파일 읽기는 주기가 짧은 반복참조가 대부분을 차지하며, 일부 임의참조가 혼합된 형태를 띄었다. 그림 3(d)의 결과파일 쓰기는 1회성 순차참조 형태를 나타내었다.

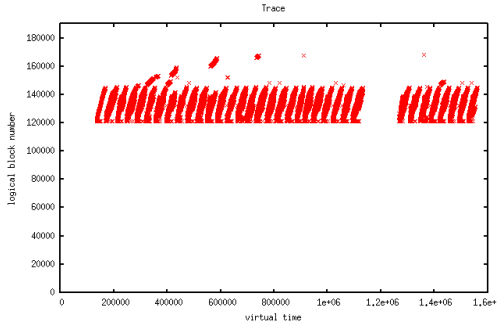
이러한 참조 패턴은 파일의 형식을 통해 파악 가능하므로 본 논문은 파일의 참조패턴에 기반해 렌더링에 특화된 버퍼캐쉬 관리기법을 설계하고자 한다. 한편, 다양한 참조패턴이 섞여있는 경우 버퍼캐쉬 공간을 분할하여 그에 적합한 관리 기법을 운용하는 것이 효과적이다. 본 논문은 렌더링 수행시 나타나는 참조패턴별 성능개선효과에 기반하여 캐쉬 공간을 할당하고 적절한 관리기법을 사용한다. 먼저, 1회성 참조 블록은 성능개선 효과가 없으므로 캐싱하지 않고, 주기가 긴 반복 참조, 주기가 짧은 반복 참조는 캐싱을 통한 이득에 기반하여 공간 할당을 한다. 이 때, 기존 연구[4]처럼 파일의 접근 패턴을 실시간으로 탐지하는 경우 그 오버헤드가 크기 때문에 본 논문은 미리 파악된 렌더링의 파일 접근 특성에 기반한 캐쉬 공간 할당 및 관리를 수행한다.



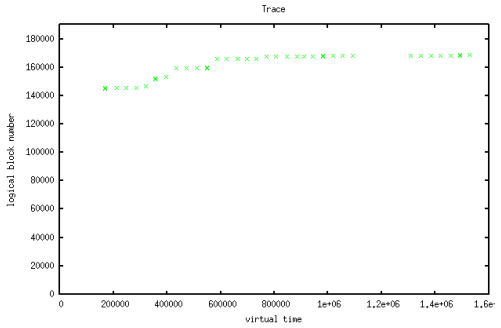
(a) 렌더링 수행 시의 전체 블록 참조



(b) 환경설정파일 참조



(c) 텍스처 입력파일 참조



(d) 결과물 파일 참조

그림 3. 렌더링의 블록 참조 패턴
Fig. 3. Block reference patterns in rendering process.

그림 4는 제안하는 버퍼캐쉬의 구조를 보여주고 있다. 캐쉬공간은 주기가 긴 환경설정영역(C1)과 주기가 짧은 텍스처영역(T1), 그리고 임의참조영역(O영역)으로 구성되며, 결과파일의 경우 재사용되지 않으므로 공간할당을 하지 않는다. 할당된 영역은 성능 기여도에 따라 크기를 조절한다.

환경설정영역과 텍스처영역의 성능기여도 확인을 위해 본 논문에서는 각각에 가상영역을 설정한다. 가상영역이란 공간 제약으로 캐쉬에서 쫓겨난 데이터에 대해 최근에 쫓겨났다는 정보를 일정 기간 유지하여 캐쉬 크기를 증가시켰을 때 어느 정도의 성능 개선을 기대할 수 있는지를 보여준다. 예를 들어 환경설정 가상영역(C2)에서 빈번한 참조가 발생한 경우 환경설정영역(C1)의 크기를 증가시켜 성능을 개선시킬 수 있다. 가상영역은 블록의 메타정보만을 유지하므로 공간 오버헤드가 매우 적은 것으로 알려져 있다[7].

제안하는 기법에서는 새로운 블록이 캐쉬에 진입하고자 할 때, 참조된 블록의 파일 형식을 확인한다. 결과파일의 경우 캐싱하지 않고, 그 외에 파일은 확장자에 기반하여 텍스처영역(T1) 또는 환경설정영역(C1)에 저장한다. 한편, T1과 C1에서의 블록 참조는 반복참조 형태이므로 블록 교체가 필요한 경우 이에 최적인 MRU(most recently used) 알고리즘을 사용하여 가장 최근에 참조된 블록을 방출한다. 한편, 방출된 블록이 반복참조가 아닌 임의참조 특성을 가진 블록일 경우 이러한 조기 방출이 성능 저하를 발생시킬 수 있으므로 본 논문에서는 곧바로 캐쉬에서 삭제하는 대신 임의참조영역인 O영역에 블록을 진입시키고, 반복참조영역의 가상영역에 그 메타정보를 남겨둔다. O영역에서 블록 교체가 필요한 경우 LRU(least recently used) 알고리즘으로 방출할 블록을 선정한다. 이는 규칙성이 없는 참조패턴에서 좋은 성능을 나타내는 가장 일반적인 알고리즘이 LRU이기 때문이다. 한편, 가상영역에도 크기 제한이 있으므로 새로운 메타정보를 가상영역에 저장시 공간부족이 발생하면 가장 오래전에 가상영역으로 진입한 블록의 메타정보를 삭제한다.

표 1 실험에 사용된 트레이스의 특성
Table 1. Characteristics of traces used in experiments.

트레이스	총 참조횟수	총 블록수
caminandes	1,562,471	168,515
dweep	1,939,659	576,752
classroom	3,308,248	361,951
wallslam	8,876,286	356,598

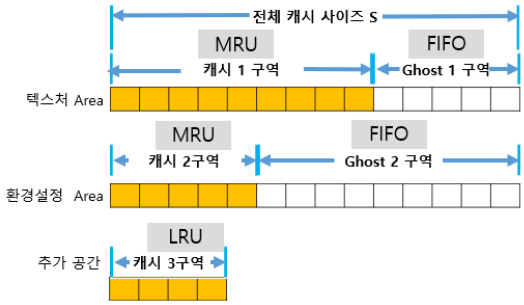


그림 4. 제안하는 버퍼캐쉬 공간의 구조
Fig. 4. The structure of the proposed buffer cache.

요청된 블록이 이미 캐쉬(또는 가상영역)에 존재하는 경우는 크게 3가지로 분류해 볼 수 있다. 그 첫째는 T1 또는 C1 구역에 존재하는 경우이다. 이 경우에는 반복참조의 루프가 캐쉬에 저장되어 캐쉬 적중이 발생한 경우로서 캐쉬 관리를 위한 메타정보의 수정은 필요하지 않다. 두번째 경우는 요청 블록이 O영역에 존재하는 경우로서 이때에는 해당 블록을 LRU 리스트 상의 가장 우선순위가 높은 지점으로 이동시킨다. 세번째 경우는 요청 블록의 정보가 가상영역에서 발견되는 경우이다. (참고로 이는 두번째 경우와 중복해서 발생할 수 있다.) 이 경우 해당 가상영역의 크기를 1 감소시키고, 해당 실제영역의 크기를 1 증가시킨다. 이때, 전체 캐쉬의 크기를 유지하기 위해 또다른 반복참조영역의 크기를 1씩 조절한다. 예를 들어 T2(또는 C2) 가상영역에서 요청 블록이 발견된 경우 T1(또는 C1)의 크기를 1증가시키는 대신, C1(또는 T1)의 크기를 1 감소시킨다. 이에 따라 C2(또는 T2) 가상영역의 크기 또한 1 증가시킨다.

Ⅳ. 실험 및 결과

제안하는 기법의 성능 평가를 위해 실제 렌더링 트레이스를 추출하고 이에 기반한 시뮬레이션 실험을 수행하였다. 트레이스 추출은 오픈소스 렌더링 소프트웨어인 Blender에서 제공하는 작업 파일 중 4가지를 무작위로 선정하였다. 표 1은 추출된 트레이스의 특성을 보여주고 있다. 제안하는 기법과의 성능 비교에는 LRU, LRU + bypass, MRU, UBM을 사용하였다. LRU + bypass는 1회성 참조인 결과파일을 캐싱하지 않고 필터링하는 방법을 사용하고 교체 알고리즘은 LRU를 적용한 기법을 뜻한다. 제안하는 기법에서 텍스처영역과 환경설정영역의

초기 비율은 워크로드 특성을 반영하여 9:1로 설정하였으며, 기타참조영역은 64KB으로 설정하였다. 이러한 크기는 워크로드의 변화로 인한 영역별 기여도 변동시 적응적으로 변화하게 된다.

그림 5는 표 1에서 설명한 4개의 트레이스에 대해 버퍼캐쉬의 크기 변화에 따른 제안 기법과 다른 기법들의 캐쉬적중률을 보여주고 있다. 그림에서 보는 것처럼 거의 모든 경우에 있어 제안하는 기법이 기존 기법들보다 우수한 성능을 나타내었다.

LRU의 경우 반복참조가 많은 렌더링 워크로드에서 루프 전체를 담을 만큼의 캐쉬 크기가 주어지지 않는 경우 재참조 시점에 블록들이 이미 캐쉬에서 쫓겨난 상태가 되기 때문에 매우 낮은 캐쉬적중률을 나타내었다. LRU가 좋은 성능을 나타내는 경우는 루프 전체를 수용할 수 있는 정도의 버퍼캐쉬 크기가 확보되는 경우에 한정된다. 따라서 대부분 캐쉬 크기 증가에 따른 성능 개선이 나타나지 않았다. LRU+bypass는 LRU보다 좋은 성능을 나타내었다. 특히, 불필요한 1회성 참조의 캐쉬 진입을 막기 때문에 유효 캐쉬 공간을 어느 정도 증가시킨 효과가 발생하여 성능이 급격히 개선되는 지점이 LRU보다 더 빠른 것을 확인할 수 있다. 그러나 LRU+bypass 역시 반복참조를 고려하지 않기 때문에 큰 성능개선 효과는 나타나지 않는 것을 확인할 수 있다.

MRU는 여러 기법 중 가장 좋지 않은 성능을 나타내었다. 실험에 사용된 트레이스는 주기가 긴 반복참조 이후에 주기가 짧은 반복참조가 나타나는 특성을 가진다. 그런데 주기가 짧은 반복참조가 전체 참조에서 차지하는 비중이 트레이스별로 49%, 82%, 89%, 95%에 이르기 때문에, MRU를 사용할 경우 이러한 블록을 캐싱하기 전에 이미 버퍼 캐쉬가 긴 주기의 반복참조에 의해 포화상태에 이르러 성능이 크게 저하되는 것을 확인할 수 있다.

UBM은 1회성 참조블록들을 캐쉬에서 빠르게 제거하면서 참조패턴별 성능개선 효과에 기반해서 공간 할당을 하기 때문에 임의참조 블록들과 주기가 짧은 반복참조부터 점진적으로 캐싱을 하여 좋은 성능을 나타내는 것을 확인할 수 있다. 하지만 이러한 성능 유지를 위해 지속적으로 참조패턴을 탐지하고 성능개선 효과를 계산하는 오버헤드가 발생한다는 점이 본 논문이 제안하는 방법과 차이점이라 할 수 있다.

본 논문이 제안한 기법은 가장 널리 사용되는 버퍼캐쉬 관리 기법인 LRU에 비해 평균 19%, 최대 55%향상된

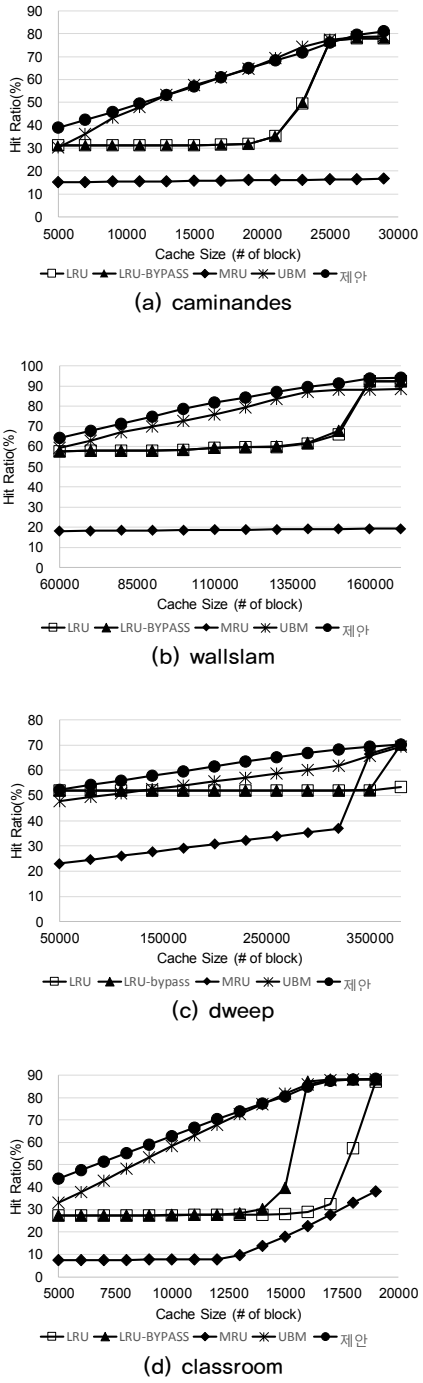


그림 5. 기존 기법과의 성능 비교
Fig. 5. Performance comparison with other schemes.

성능을 나타내었으며, LRU+bypass 대비 평균 15%, 최대 47% 성능 개선을 나타내었다. 또한, MRU 대비 평균 45%, 최대 74%, UBM 대비 평균 3%, 최대 10% 향상된 성능을 나타내었다.

V. 결 론

본 논문에서는 렌더링 시스템에 특화된 버퍼캐쉬 관리 기법을 설계하였다. 일반적인 목적의 시스템과 달리 렌더링 시스템에서는 고유한 파일 입출력 특성이 나타나며, 이로 인해 기존의 버퍼캐쉬에서 렌더링 작업 수행시 큰 성능저하가 발생함을 분석하였다. 이를 개선하기 위해서 본 논문에서는 렌더링 파일 참조패턴이 반복 참조, 1회성참조, 임의참조 등으로 구성된다는 점을 분석하고, 각 참조패턴에 기반한 캐쉬 공간 할당 및 관리 방안을 제안하여 기존 기법 대비 평균 19%, 최대 55% 성능 개선을 달성하였다.

References

[1] H. Yoon and J. Park, “Non-photorealistic rendering of mixed contents of 3D models and video footage,” Journal of KIIT, Vol.12, No.7, pp.199-206, 2014.
DOI: http://10.14801/kiitr.2014.12.7.199

[2] J. Kim, “A study on the photorealism of digital architectural rendering images,” Journal of the Korea Academia-Industrial cooperation Society (JKAIS), Vol. 19, No. 2, pp. 238-246, 2018.
DOI: https://doi.org/10.5762/KAIS.2018.19.2.238

[3] D. Kim and H. Bahn, “Buffer cache management of smartphones exploiting write-only-once characteristics,” The Journal of the Institute of Internet, Broadcasting and Communication (IIBC) Vol. 15, No. 6, pp.129-134, 2015.
DOI: http://doi.org/10.7236/JIIBC.2015.15.6.129

[4] J. Kim et al., “A low-overhead high-performance unified buffer management scheme that exploits sequential and looping references,” Proc. OSDI Conf., 2000.

[5] N. Megiddo and D. Modha, “ARC: a self-tuning, low overhead replacement cache,” Proc. USENIX FAST Conf., Vol. 3, pp. 115-130, 2003.

[6] H. Lee and H. Bahn, “Characterizing virtual memory write references for efficient page replacement in NAND flash memory,” Proc. IEEE MASCOTS Conf., pp. 1-10, 2009.
DOI: https://doi.org/10.1109/mascot.2009.5366768

[7] T. Johnson and D. Shasha, “2Q: a low overhead high performance buffer management replacement algorithm,” Proc. VLDB Conf., 1994.

저자 소개

신 동 희(회원)

• 2016년 8월 : 이화여자대학교 교육공학과 학사
• 2017년 3월 ~ : 이화여자대학교 컴퓨터공학과 석사과정
<주관심분야 : 운영체제, 스토리지 시스템>

반 효 경(정회원)

• 1997년 2월 : 서울대학교 계산통계학과 학사
• 1999년 2월 : 서울대학교 전산과학과 석사
• 2002년 2월 : 서울대학교 컴퓨터공학부 박사.
• 2002년 9월 ~ : 이화여자대학교 컴퓨터공학과 교수
<주관심분야 : 운영체제, 스토리지시스템, 임베디드시스템>

※ This work was supported by the ICT R&D program of MSIP/IITP (2018-0-00549, Extremely Scalable Order-preserving Operating System for Manycore and Non-volatile Memory) and also by the Basic Science Research program through NRF grant funded by the Korea government (MSIP) (No. 2016R1A2B4015750).