논문 2022-17-08

멀티코어 시스템에서 TLB Lockdown에 의한 TLB Miss 영향 분석

(Investigation on TLB Miss Impact through TLB Lockdown in Multi-core Systems)

송 대 영, 박 시 형, 김 형 신*
(Daeyoung Song, Sihyeong Park, Hyungshin Kim)

Abstract : Virtual memory is used as the method to ensure the safety of the system through memory protection in the real-time system. TLB miss caused by using virtual memory makes the real-time system WCET more pessimistically. TLB lockdown can be applied as a method to improve this problem. However, processors with limited TLB lockdown entries, a selection criterion is needed to efficiently utilize the TLB lockdown entry. In this paper, the most frequently accessed virtual pages in the process are applied to the TLB lockdown by analyzing memory profiling. The results showed that micro data TLB miss stall cycle and main data TLB miss stall cycle of the processor decreased by at least 4.7% and up to 29.7%.

Keywords : TLB miss, TLB lockdown, Memory profiling, Multi-core system

Ⅰ. 서 론

현대의 범용 운영체제는 효율적인 메모리 관리를 위해서 가상 메모리를 사용한다. 하지만 가상 메모리를 사용함으로써 발생하는 지연 시간과 확률적 특성은 프로세스의 실행 시간을 예측하기 어렵게 한다. 특히, 프로세스의 실행 시간을 보장해야 하는 실시간 시스템에서는 최악 실행 시간(Worst-Case Execution Time)을 측정하는 것이 중요하다 [1]. 따라서 실시간 시스템에서는 태스크의 실행 시간 예측성을 높이기 위해 가상 메모리를 사용하지 않는다 [2].

실시간 시스템의 실시간성 저하에도 불구하고 시스템의 안전성을 보장하는 방법으로 가상 메모리를 사용한다 [3, 4]. 가상 메모리는 프로세스 간 메모리 영역을 분할하여 잘못된 주소 접근으로 인한 오류를 예방하고 프로세스에서 발생하는 오류를 격리하여 시스템의 안정성을 보장한다 [5].

가상 메모리를 사용하는 시스템은 가상 주소를 물리 주소로 빠르게 변환하기 위해 TLB (Translation Look-aside Buffer)를 사용한다 [6]. 프로세서는 가상 주소를 물리 주소로 변환할 때 가장 먼저 TLB에 접근한다. 최근 사용한 주소 정보를 저장하는 TLB는 가상 주소를 물리 주소로 변환할 때 메인 메모리에 위치한 페이지 테이블 접근 횟수를 감소시킨다. 만약 TLB에 요청한 주소 정보가 없는 경우 TLB miss가 발생한다. TLB miss가 발생하면, 프로세서는 메인 메모리에 위치한 페이지 테이블에 접근하여 해당 정보를 검색하는 페이지 테이블 워크를 수행한다. 페이지 테이블 워크는 교체되는 TLB 엔트리를 예측하기 어렵고 지연 시간이 발생하기에, 시스템의 최악 실행 시간 예측성을 떨어뜨린다.

TLB miss 발생을 감소시키는 방법으로 TLB lockdown이 제안되었다 [7]. TLB lockdown은 지정된 TLB 엔트리를 TLB 교체 정책에서 제외하는 방법이다. 이를 통해서, 태스크가 접근하는 페이지를 lockdown을 적용하여 TLB miss로 인한 프로세스의 최악 실행 시간을 개선할 수 있다. 기존 TLB lockdown을 적용한 연구에서는 태스크가 사용하는 모든 페이지를 TLB lockdown 하여 태스크의 최악 실행 시간을 개선하였다. 하지만, 해당 연구는 제한된 TLB lockdown 엔트리를 가지는 싱글코어 프로세서에서 진행되었다. 따라서, TLB lockdown 엔트리가 제한된 프로세서와 멀티코어 프로세서 환경에서는 적용할 수 없다.

본 논문에서는 가상 메모리를 사용하는 실시간 시스템의 최악 실행 시간을 감소하기 위한 방법으로 TLB lockdown을 적용하였다. 제한된 TLB lockdown 엔트리를 가지는 멀티코어 프로세서를 사용하였으며, 이중 시스템에서 가장 빈번하게 접근하는 페이지를 lockdown 기준으로 선택하였다. TLB lockdown을 통한 TLB miss 발생을 확인하기 위해 ARM Cortex-A9 프로세서 기반의 Sabre Lite 보드를 사용하였다. 해당 보드는 쿼드코어 프로세서를 탑재하였고 128개의 main TLB 엔트리 중 4개의 data TLB lockdown 엔트리를 제공한다. 프로세스에서 가장 빈번하게 접근하는 페이지를 찾기 위해 메모리 프로파일링 도구를 사용하였다. 메모리 프로파일링을 활용하여 얻은 프로세스 메모리 접근 정보를 분석하여 가장 빈번하게 접근하는 페이지를 도출하였으며, 해당 페이지들을 lockdown 하였다. 멀티코어 환경에

*Corresponding Author (hyungshin@cnu.ac.kr)
Received: Sep. 30, 2021, Revised: Nov. 13, 2021, Accepted: Feb. 8, 2022.
D.Y. Song: Suresoft Tech Co., Ltd (Senior Researcher)
S.H. Park: KETI (Senior Researcher)
H.S. Kim: Chungnam National University (Prof.)

ⓒ IEMEK J. Embed. Sys. Appl. 2022 Feb. 17(1) 59-65
ISSN : 1975-5066
http://dx.doi.org/10.14372/IEMEK.2022.17.1.59

서는 페이지 테이블과 코어 내 TLB 간의 동기화 오버헤드가 싱글코어 환경보다 크기에 실험 환경이 다르다. 따라서, 싱글코어 환경뿐만 아니라 멀티코어 환경에서도 TLB lockdown 영향을 확인하기 위한 실험 시나리오를 구성하였다. 실험 결과 micro Data TLB cycle은 최소 5.1% 최대 29.7% 감소하였으며, main Data TLB cycle은 최소 4.7% 최대 7.6% 감소하였다.

본 논문의 구성은 다음과 같다. 2장에서는 TLB miss 발생을 감소시키기 위해서 진행된 관련 연구를 소개한다. 3장에서는 본 논문에서 사용한 ARM Cortex-A9 기반 프로세서의 TLB 구조와 TLB miss 처리 절차에 대하여 설명한다. 4장에서는 메모리 프로파일링을 활용하여 얻은 메모리 접근 정보를 분석하여, 프로세스에서 가장 빈번하게 접근하는 페이지 번호를 도출하는 방법을 설명한다. 5장에서는 실험 시나리오를 구성하고 TLB lockdown 적용에 따른 프로세서의 TLB miss 대기 사이클을 측정 후 분석한다. 6장에서는 결론과 향후 연구를 기술한다.

Ⅱ. 관련 연구

TLB lockdown은 하드웨어의 지원이 필요한 기능으로, 본 논문에서 사용하는 ARM Cortex-A9 기반 프로세서를 포함한 일부 상용 아키텍처에서 해당 기능을 지원한다 [8]. 이 중 SuperH 아키텍처 기반 프로세서에서 제공하는 TLB lockdown을 적용하여, 태스크의 최악 실행 시간을 개선하였다 [7]. 태스크에서 사용하는 모든 페이지를 lockdown 하여, TLB miss가 발생하지 않도록 하였다. 하지만 제한된 TLB lockdown 엔트리를 가지는 싱글 프로세서만을 고려하였다.

TLB lockdown을 가상화 환경의 모드 전환에 적용한 연구가 있다 [9]. 해당 연구에서는 Xen 하이퍼바이저에서 특권 모드와 비특권 모드의 전환 시간을 개선하기 위해서 TLB lockdown을 적용하였다. 8개의 TLB lockdown 엔트리를 제공하는 프로세서를 사용하였으며, 이 중 Xen 하이퍼바이저가 사용하는 두 개의 페이지를 lockdown 하였다.

태스크의 힙 메모리에서 발생하는 TLB miss를 감소시키는 방법으로 TLB coloring [10]이 제안되었다. TLB coloring은 프로세스가 힙 메모리를 할당할 때 각 프로세스에서 사용하는 페이지에 독립된 색 (color)을 칠함으로써, 프로세스 간 data TLB 영역을 분할하는 방법이다. 각 프로세스에 색칠된 data TLB는 다른 data TLB로 교체되지 않고 페이지 정보를 유지한다. 따라서, 프로세스 간 그리고 프로세스 내 간섭으로 TLB miss가 발생하지 않는다. 하지만 비결정적 시간이 소요되는 힙 메모리 할당은 최악 실행 시간 예측성을 떨어뜨리므로, 실시간 시스템에서 TLB coloring을 적용하기 어렵다 [11].

시스템에서 발생하는 TLB miss를 감소시키기 위해서 슈퍼페이지를 사용할 수 있다 [12, 13]. 가상 메모리의 페이지 크기는 프로세서에서 지원하는 페이지 크기에 결정되며, 일반적으로 4KB 크기의 페이지를 사용한다. 슈퍼페이지는 일반적으로 사용하는 4KB 크기의 페이지보다 큰 페이지 크기를 사용한다. 슈퍼페이지는 하나의 TLB 엔트리에 매핑되는 가상 메모리 영역을 증가시킴으로써 TLB miss 발생을 감소시킨다. 하지만 시스템에서 사용하는 페이지 크기가 커질수록 페이지의 메모리 내부 단편화로 메모리 공간이 낭비된다. 이러한 문제를 해결하기 위해서, 프로세스가 사용하는 메모리 크기를 예측한 다음 슈퍼페이지와 일반 페이지를 동적으로 할당하는 기법이 제안되었다 [13].

Ⅲ. ARM Cortex-A9 기반 프로세서의 TLB 구조

그림 1은 본 논문에서 사용한 ARM Cortex-A9 프로세서를 탑재한 Sabre Lite 보드의 TLB 구조다. ARM Cortex-A9 기반 프로세서는 micro TLB와 main TLB 두 개의 계층으로 구성된다 [14]. 첫 번째 계층인 micro TLB는 각 32개의 엔트리를 가지는 instruciton TLB와 data TLB로 구성된다. 프로세서는 가장 먼저 각 코어의 micro TLB에 접근하여 요청한 주소 정보를 검색하며, 해당 과정에는 단일 사이클이 소요된다. 만약 프로세서가 micro TLB에서 검색에 실패한다면, main TLB에 접근하여 검색한다. Main TLB는 data TLB와 instruction TLB를 하나로 통합한 구조이며, 총 128개의 엔트리로 구성된다. Cortex-A9 기반 프로세서는 4개의 lockdown 가능한 data main TLB 엔트리를 제공한다. Main TLB는 micro TLB 보다 페이지 주소 검색 속도가 느리며, 외부 환경 요소에 의하여 검색에 소요되는 사이클 수가 가변 된다. 만약 main TLB miss가 발생하면, 프로세서는 하드웨어 단계에서 페이지 테이블 워크를 수행한다.

그림 2는 페이지 테이블 워크 처리 절차다. 프로세서는 메인 메모리에 위치한 페이지 테이블에 접근한 다음 요청한 주소 정보를 검색한다. 페이지 테이블에 주소 정보가 존재하는 경우 (페이지 테이블 hit), 해당 주소 정보를 프로세서에 반환한다. 페이지 테이블 워크에 따른 메인 메모리 접근

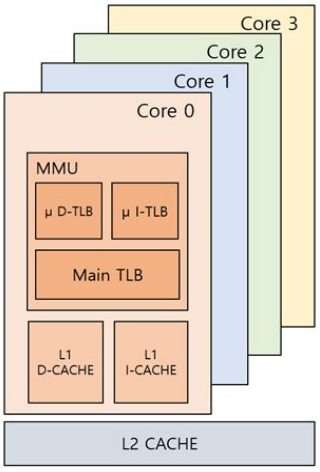

그림 1. ARM Cortex-A9 TLB 구조
Fig. 1. ARM Cortex-A9 TLB organization

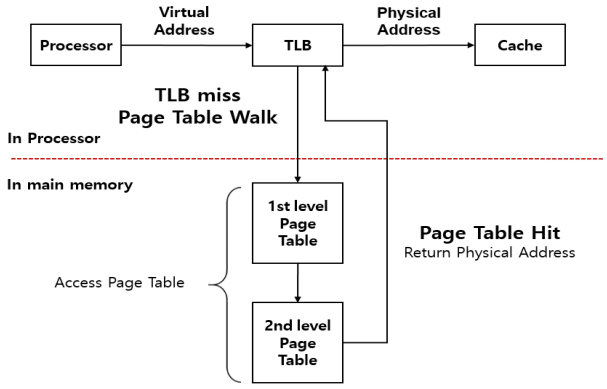

그림 2. 페이지 테이블 워크 처리 절차
Fig. 2. Page table walk handling process

횟수는 시스템에서 사용하는 페이지 테이블 단계에 비례하여 증가한다. 메모리 접근 횟수가 증가함에 따라 페이지 테이블 워크로 인한 지연 시간도 증가한다 [15]. 2단계 페이지 테이블 구조를 가지는 ARM Cortex-A9 기반 프로세서는 2번의 연속적인 페이지 테이블 접근이 발생한다. TLB miss는 약 0.01~2% 확률로 발생하며, TLB miss 페널티에 약 10~1,000 사이클이 소요된다 [16].

프로세서는 운영체제에서 관리하는 페이지 테이블과 각 코어의 TLB 간의 일관된 가상 주소를 유지하기 위한 동기화 방법으로 TLB shootdown을 수행한다 [17]. 메인 메모리에 있는 페이지 테이블 정보가 변경되거나 다른 태스크로 문맥 전환이 발생할 때 TLB shootdown이 수행되며, 운영체제는 프로세서를 강제로 중단하고 각 코어에서 유효하지 않은 모든 TLB를 invalidate 함으로서 TLB의 일관성을 유지한다. 멀티코어 환경에서 사용하는 코어의 개수가 증가할수록 TLB shootdown의 비용과 발생 빈도가 증가하여 시스템 성능이 저하된다. 본 논문에서 사용하는 ARM Cortex-A9 프로세서는 TLB invalidate 명령을 통하여 각 코어의 micro TLB와 main TLB를 invalidate 한다.

Ⅳ. 메모리 프로파일링

메모리 프로파일링 전 리눅스의 ASLR (Address Space Layout Randomization) 보안 기능을 비활성화하였다. ASLR은 프로세스가 실행될 때마다 메모리 주소를 임의로 변경하는 기능으로, 프로세스의 메모리 취약점을 보완한다 [18]. 이를 통해서, 프로세스가 항상 같은 메모리 주소를 할당받도록 하였으며, 페이지가 같은 주소에 lockdown 되도록 보장하였다. 이를 통해서, 프로세스가 항상 같은 메모리 주소를 할당받도록 하였으며, 페이지가 같은 주소에 lockdown 되도록 보장하였다.

프로세스가 접근하는 페이지들을 도출하기 위해 메모리 프로파일링 도구인 Valgrind Lackey를 사용하였다 [19]. Valgrind Lackey 도구는 프로세스의 메모리 접근 정보를 수집하는 기능을 제공한다. 해당 도구를 사용하여 프로세스의 data 메모리 접근 정보를 수집하였다. 이 중 가상 메모리 주소의 12~31bit에 해당하는 페이지 번호를 접근 횟수에 따라 분류하였다. 해당 과정을 통하여, 프로세스에서 가장 빈번하게 접근하는 4개의 페이지 번호를 도출하였다.

수집한 프로세스의 메모리 접근 정보와 프로세스에 할당된 메모리 영역을 비교하여, 일반 실행환경에서 실행되는 프로세스에서 가장 빈번하게 접근하는 페이지를 도출하였다. 메모리 프로파일링 환경은 Valgrind Lackey가 사용하는 메모리 영역으로 인하여 일반 실행 환경보다 더 많은 메모리를 사용한다. 프로세스의 메모리 주소 구조가 다르며, 각

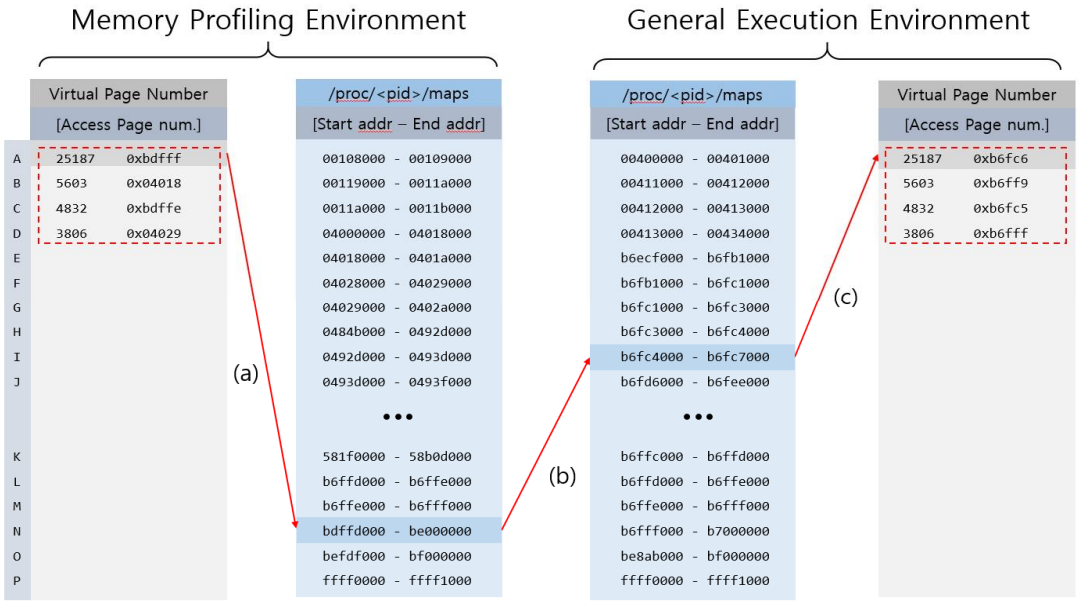

그림 3. 페이지 번호 매핑 과정
Fig. 3. Virtual page number mapping process

영역의 메모리 시작 주소와 사용하는 메모리 주소 범위가 변경된다. 따라서, 각 환경에서의 프로세스 메모리 영역을 비교하여 일반 실행 환경에서 가장 빈번하게 접근하는 페이지를 매핑하는 과정이 필요하다. 해당 과정에는 프로세스에 할당된 메모리 영역이 기록되는 /proc/<pid>maps 파일 사용하였다 [20].

그림 3은 메모리 프로파일링 환경에서 수집한 메모리접근 정보를 바탕으로 가장 빈번하게 접근한 페이지 번호를 일반 실행 환경으로 매핑하는 과정이다. 프로세스에 할당된 메모리 영역 범위는 [시작 주소 - 끝 주소]로 표현된다. 메모리 프로파일링 환경에서 가장 빈번하게 접근하는 페이지 번호를 프로세스에 할당된 메모리 영역으로 매핑하였다. A 번째 줄의 0xbdfff 페이지 번호는 N 번째 줄의 0xbdffd000 - 0xbe000000 범위로 매핑된다 (a). 메모리 프로파일링 환경의 프로세스 메모리 영역을 일반 실행 환경의 프로세스 메모리 영역으로 매핑하였다. 메모리 프로파일링 환경의 M 번째 줄인 0xbdffd000 - 0xbe000000 범위는 일반 실행 환경의 I 번째 줄인 0xb6fc4000 - b6fc7000 범위로 매핑된다 (b). 일반 실행 환경의 가상 메모리 영역이 가상 페이지 번호로 매핑하기 위해서는, 메모리 프로파일링 환경의 가상 메모리 주소와 가상 페이지 번호의 차이를 계산한 오프셋이 필요하다. 일반 실행 환경의 I 번째 줄인 0xbdfc4000 - 0xb6fc7000 범위의 시작 주소인 0xb6fc4에 메모리 프로파일링에서 계산한 오프셋인 0x00002를 더하여 0xb6fc6 페이지 번호를 도출하였다 (c). 해당 과정을 통하여 일반 실행 환경에서 가장 빈번하게 접근하는 4개의 페이지 번호를 도출하였으며, 해당 페이지들을 TLB lockdown에 적용하였다.

V. 실험 및 결과

1. 실험 환경 및 시나리오

실험에는 ARM Cortex-A9 기반 프로세서를 탑재한 SABRE Lite (i.MX6) 보드를 사용하였다. 해당 보드는 792MHz로 동작하는 쿼드코어 프로세서를 탑재하였다. 이를 위해 RREEMPT_RT 패치가 적용된 리눅스 커널 4.14 버전과 랜덤 TLB 교체 정책을 사용하였다.

TLB lockdown을 적용할 때의 성능을 측정하기 위해서 프로세서의 Performance Monitoring Unit (PMU)를 사용하였다 [7, 14]. PMU는 시스템 실행 중에 내부 작동에 대한 이벤트를 수집하는 기능을 제공한다. 따라서, 본 논문에서는 micro data TLB miss와 main data TLB miss 발생으로 인한 프로세서의 대기 사이클 (TLB miss stall cycle) 측정 이벤트를 설정하여, 기록된 값을 읽어와서 성능 측정하였다.

실험 시나리오를 구성하기 위해 MiBench 벤치마크를 사용하였다. MiBench 벤치마크는 오픈소스 임베디드시스템 벤치마크로써 사용 목적에 따라 6개의 카테고리를 제공한다 [21]. 이 중 'Automotive and Industrial' 카테고리와 'Consumer' 카테고리를 사용하여 실험 시나리오를 구성하였다. 각 카테고리에서 사용한 벤치마크에 대한 설명은 표 1

표 1. 실험에서 사용한 벤치마크
Table 1. Benchmarks used in experiments

Category	Benchmark	Explanation
Automotive and Industrial (Real-time Process)	basicmath	Mathematical calculations
	qsort	Sort array of string
	susan	Recognize image corner and edge
Consumer (Non real-time process)	cjpeg	Compress image
	djpeg	Decompression image
	typeset	Convert text to electronic document format

과 같다. 'Automotive and Industrial' 카테고리는 실시간 스케줄링 정책인 SCHED_FIFO를 적용하였다. 'Consumer' 카테고리는 비실시간 스케줄링 정책인 SCHED_OTHER을 적용하였다. 실시간 스케줄링 정책은 비실시간 스케줄링 정책보다 높은 우선순위를 보장한다.

TLB lockdown이 싱글코어 시스템과 멀티코어 시스템 모두 영향을 주는지 확인하기 위해서 각 환경별 실험 시나리오를 구성하였다. 싱글코어 환경에서는 비실시간 프로세스인 cjpeg 벤치마크와 djpeg 벤치마크를 실행한 다음 실시간 프로세스 벤치마크를 실행하였다. 하나의 코어를 제외한 다른 코어들을 모두 비활성화하였다. 멀티코어 환경에서는 모든 코어에서 프로세스가 동시에 실행되도록 구성하였다. 1번부터 3번 코어까지 비실시간 프로세스인 cjpeg 벤치마크, djpeg 벤치마크 그리고 typeset 벤치마크를 반복해서 실행하였다. 4 번째 코어는 비실시간 프로세스인 cjpeg 벤치마크와 djpeg 벤치마크를 실행한 다음 실시간 프로세스 벤치마크를 실행하였다. 각 환경의 실시간 프로세스에 대해서 100번 성능 측정하였다.

2. 실험 결과

그림 4와 그림 5는 싱글코어 환경과 멀티코어 환경에서 TLB lockdown을 적용한 실험 시나리오의 data TLB miss stall cycle을 나타낸 그래프다.

TLB lockdown을 적용한 경우 싱글코어 환경에서의 micro data TLB stall cycle은 최소 5.1% 최대 29.7% 감소하였으며, main data TLB stall cycle은 최소 4.7% 최대 7.6% 감소하였다. 멀티코어 환경에서의 micro data TLB stall cycle은 최소 4.7% 최대 23.3% 감소하였으며, main data TLB stall cycle은 최소 2.9% 최대 7.6% 감소하였다. 이를 통해서, 메모리 프로파일링으로 도출한 프로세스에서 가장 빈번하게 접근하는 페이지를 lockdown 하는 경우, TLB miss stall cycle이 감소한 것을 확인하였다.

표 2와 표 3은 각 싱글코어 환경과 멀티코어 환경에서 기존 실험 시나리오와 TLB lockdown을 적용한 실험 시나리오의 data TLB miss stall cycle 차이를 계산한 값이다. 싱글코어 환경과 멀티코어 환경 모두 basicmath 벤치마크, susan 벤치마크 그리고 qsort 벤치마크 순서대로 감소한 micro data TLB stall cycle이 크게 측정되었다. 감소한 main data TLB miss stall cycle이 micro data TLB miss

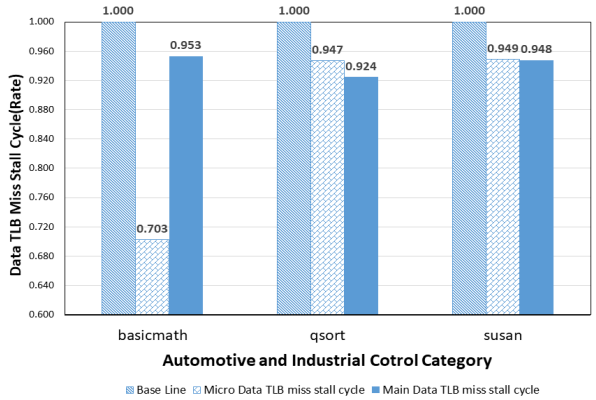

그림 4. 싱글코어 환경에서의 data TLB miss stall cycle
Fig. 4. Data TLB miss stall cycle in single-core

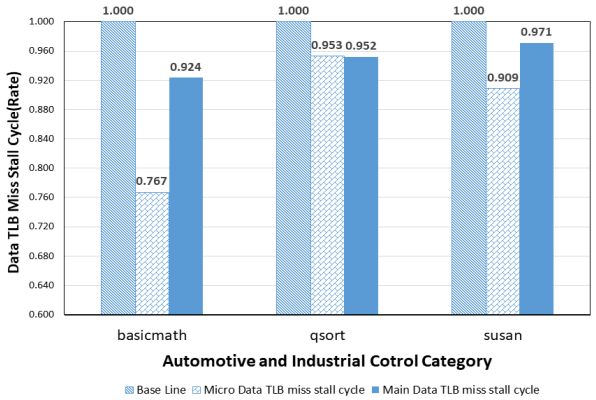

그림 5. 멀티코어 환경에서 data TLB miss stall cycle
Fig. 5. Data TLB miss stall cycle in multi-core

stall cycle보다 크게 측정되었다.

연산 위주의 basicmath 벤치마크의 경우 메모리 접근 횟수가 적기에, TLB miss stall cycle이 적게 측정되었다. 따라서, TLB lockdown을 적용하면 감소한 data TLB miss stall cycle 또한 적게 측정되었다. 하지만, qsort 벤치마크와 susan 벤치마크는 basicmath 벤치마크보다 메모리에 빈번하게 접근하기에, TLB miss stall cycle이 크게 측정되었다. 따라서, TLB lockdown을 적용하면 감소한 data TLB miss stall cycle 도 크게 측정되었다. qsort 벤치마크는 많은 문자열 배열을 사용하며 다른 페이지를 호출하는 횟수가 더 빈번하기에, susan 벤치마크보다 감소한 data TLB miss stall cycle이 더 크게 측정되었다. 실험 결과 data TLB miss가 많이 발생하는 프로세스에 TLB lockdown을 적용하는 경우, 감소하는 data TLB miss stall cycle이 증가하였다.

basicmath 벤치마크 수행에 약 1500만 사이클, qsort 벤치마크 수행에 약 3700만 사이클, susan 벤치마크 수행에 약 3300만 사이클이 소요되었다.

페이지 호출을 적게하는 basicmath 벤치마크의 경우 TLB miss stall cycle이 전체 수행 사이클의 약 0.0001%를 차지하며, 페이지 호출을 많이하는 qsort 벤치마크의 경우 TLB miss stall cycle이 전체 수행 사이클의 약 0.005%를 차지한다. 본 논문의 실험 결과를 통해 제안한 TLB lockdown 방법이 실행 시간에 미치는 영향은 미미하지만, 실시간 시스템의 관점에서 TLB miss stall cycle을 줄여 실행시간의 예측성을 개선할 수 있다. 이를 통해, 최악 실행 시간이 감소됨을 보였다.

표 2. 싱글코어 환경에서 감소한 data TLB miss stall cycle
Table 2. Decreased data TLB miss stall cycle in single-core environment

Benchmark	Decreased micro data TLB miss stall cycle	Decreased main data TLB miss stall cycle
basicmath	127	75
qsort	4,011	9,683
susan	150	874

표 3. 멀티코어 환경에서 감소한 data TLB miss stall cycle
Table 3. Decreased data TLB miss stall cycle in multi-core environment

Benchmark	Decreased micro data TLB miss stall cycle	Decreased main data TLB miss stall cycle
basicmath	109	210
qsort	3,560	7,361
susan	290	651

Ⅵ. 결론 및 향후 연구

가상 메모리를 사용하는 실시간 시스템에서 TLB miss 발생은 최악 실행 시간 예측성을 떨어뜨린다. 이러한 문제점을 개선하기 위해서 TLB lockdown이 제안되었다. 하지만 제한된 TLB lockdown 엔트리 개수가 있는 멀티코어 프로세스에서는 연구가 진행되지 않았다. 본 논문에서는 프로세서의 제한된 TLB lockdown 엔트리를 효율적으로 사용할 수 있는 방법을 제안하였다. 메모리 프로파일링을 통하여 프로세스에서 가장 빈번하게 접근하는 페이지를 lockdown 하였다. 싱글코어 환경과 멀티코어 환경에서 실험 시나리오를 구성하여, TLB lockdown 성능을 측정하였다. 실험 결과 각 환경 모두에서 micro data TLB miss와 main data TLB 발생으로 인한 프로세서의 대기 사이클이 최소 4.7% 최대 29.7% 감소하였다.

본 논문의 실험 결과, 멀티코어 환경의 data TLB miss stall cycle이 싱글코어 환경보다 20% 높게 측정되었다. 멀티코어 환경의 경우 코어 간의 TLB 동기화 처리 시간이 증가하였기 때문이다. 하지만 본 실험에서는 싱글코어 환경과 멀티코어 환경에서 결과에 대한 분석이 부족하고, TLB miss가 프로세스 실행 시간에 큰 영향을 미치는 벤치마크에 대하여 추가 실험이 필요하다. 그리고 프로세스에서 빈번하게 사용하는 TLB 엔트리 개수가 많은 경우, TLB

lockdown을 적용하기 어렵다는 한계점이 있다. 향후 연구로써, TLB miss가 더 빈번하게 발생하는 프로세스를 싱글코어 환경과 멀티코어 환경에서 실험하여 결과를 분석하고자 한다.

References

[1] G. Bernat, A. Colin, S. M. Petters, "WCET Analysis of Probabilistic Hard Real-time Systems," Proc. of the 23rd IEEE Real-Time Systems Symposium, pp. 279-288, 2002.

[2] S. Chattopadhyay, Embedded System Design, 2nd Ed., PHI Learning Pvt. Ltd., 2013.

[3] M. D. Bennett, N. C. Audsley, "Predictable and Efficient Virtual Addressing for Safety-critical Real-time Systems," Proc. of the 13th Euromicro Conference on Real-Time Systems, pp. 183-190, 2001.

[4] P. Parkinson, L. Kinnan, "Safety-critical Software Development for Integrated Modular Avionics," Embedded System Engineering, Vol. 11, No. 7, pp. 40 - 41, 2003.

[5] C. Mercer, R. Rajkumar, J. Zelenka, "Temporal Protection in Real-time Operating Systems," Proc. of the 11th IEEE Workshop on Real-Time Operating Systems and Software, pp. 79-83, 1994.

[6] S. Mittal, "A Survey of Techniques for Architecting TLBs," Concurrency and Computation: Practice and Experience, Vol. 29, No. 10, 2017.

[7] T. Ishikawa, T. Kato, S. Honda, H. Takada, "Investigation and Improvement on the Impact of TLB Misses in Real-time Systems," Proc. of OSPERT, 2013.

[8] https://static.docs.arm.com/ddi0406/c/DDI0406C_C_arm_architecture_reference_manual.pdf/.

[9] J. Y. Hwang, S. B. Suh, S. K. Heo, C. J. Park, J. M. Ryu, S. Y. Park, C. R. Kim, "Xen on ARM: System Virtualization Using Xen Hypervisor for ARM-based Secure Mobile Phones," Proc. of the 5th IEEE Consumer Communications and Networking Conference, pp. 257-261, 2008.

[10] S. A. Panchamukhi, F. Mueller, "Providing task Isolation Via TLB Coloring," Proc. of the 21st IEEE Real-Time and Embedded Technology and Applications Symposium, pp. 3-13, 2015.

[11] M. Masmano, I. Ripoll, P. Balbastre, A. Crespo, "A Constant-time Dynamic Storage Allocator for Real-time Systems," Real-Time Systems, Vol. 40, No. 2, pp. 149-179, 2008.

[12] T. H. Romer, W.H. Ohlrich, A. R. Karllin, B. N. Bershad, "Reducing TLB and Memory Overhead Using Online Superpage Promotion," Proc. of the 22nd Annual International Symposium on Computer Architecture, pp. 176-187, 1995.

[13] M. M. Papadopoulou, X. Tong, A. Seznec, A. Moshovos, "Prediction-based Superpage-friendly TLB Designs," Proc. of the 21st IEEE International Symposium on High Performance Computer Architecture, pp. 210-222, 2015.

[14] http://infocenter.arm.com/help/topic/com.arm.doc.ddi0388f/DDI0388F_cortex_a9_r2p2_trm.pdf/.

[15] Y. Li, R. Melhem, A. K. Jones, "PS-TLB: Leveraging Page Classification Information for Fast, Scalable and Efficient Translation for Future CMPs," ACM Transactions on Architecture and Code Optimization, Vol. 9, No. 4, 2013.

[16] D. A. Patterson, J. L. Hennessy, "Computer Organization and Design: The Hardware/Software Interface (The Morgan Kaufmann Series in Computer Architecture and Design)," 5th Ed., Morgan Kaufmann Publishers, 2013.

[17] N. Amit, "Optimizing the TLB Shootdown Algorithm with Page Access Tracking," USENIX Annual Technical Conference (USENIX ATC 17), pp. 27-39, 2017.

[18] C. Giuffrida, A. Kuijsten, A. S. Tanenbaum, "Enhanced Operating System Security Through Efficient and Fine-grained Address Space Randomization," Proc. of the 21st USENIX Security Symposium, pp. 475-490, 2012.

[19] A. J. Pena and P. Balaji, "A Framework for Tracking Memory Accesses in Scientific Applications," Proc. of the 43rd International Conference on Parallel Processing Workshops, pp. 235-244, 2014.

[20] http://man7.org/linux/man-pages/man7/sched.7.html/.

[21] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, R. B. Brown, "MiBench: A Free, Commercially Representative Embedded Benchmark Suite," Proc. of the Fourth Annual IEEE International Workshop on Workload Characterization, pp. 3-14, 2001.

Daeyoung Song (송 대 영)

2017 Computer Science and Engineering from Hannam University (B.S.)

2019 Computer Science and Engineering from Chungnam National University (M.S.)

Career:

2019~Senior Researcher in Suresoft Tech Co., Ltd

Field of Interests: Multi-core Embedded System, Embedded System Testing

Email: dysong@suresofttech.com

Sihyeong Park (박 시 형)

2014 Computer Science and Engineering from Chungnam National University (B.S.)
2016 Computer Science and Engineering from Chungnam National University (M.S.)
2021 Computer Science and Engineering from Chungnam National University (Ph.D.)

Career:
2021~SoC Platform Research Center from Korea Electronics Technology Institute (Senior Researcher)

Field of Interests: Multi-core Embedded System, Real-time Operating System

Email: sihyeong@keti.re.kr

Hyungshin Kim (김 형 신)

1990 Computer Science from KAIST(B.S.)
1991 Satellite Communications Engineering from Univ. of Surrey, UK(M.S.)
2003 Computer Science from KAIST(PhD)
2004~Dept. of Computer Science and Engineering, Chungnam National Univ.(Prof.)

Career:
1992 Researcher, SaTReC, KAIST
2003 Post Doc Researcher, CMU

Field of Interests: Real-time Embedded software, Embedded AI Computing

Email: hyungshin@cnu.ac.kr