Project 4: Multi-level Cache Model and Performance Analysis

Due 23:59, 11th June, 2023

TA: Minhyeok An(mh.an@dgist.ac.kr), Jungbo Kim(jbkim@dgist.ac.kr)

Sungju Kim(sungju_kim@dgist.ac.kr), YunHyeong Jeon(yhjeon@dgist.ac.kr)

Introduction

본 과제의 목표는 2 단계(Two level) 캐시 시뮬레이터를 구현 및 실험을 통해 기본적인 캐시의 구조를 이해하는 것이다.

Modeling a Two-Level Cache

1. Implementation

A. Specification

다음과 같은 사양의 2단계 캐시를 구현한다.

- Cache는 2단계 구조로 이뤄져 있고 Level 1 cache(L1)의 경우 data를 위한 L1d 하나로 구성된다. Level 2 cache(L2)는 inclusive cache로 구성된다.
- Cache replacement 정책으로 Random과 LRU(Least recently used)방식을 구현한다.
- Inclusive Cache는 다음과 같이 작동하는 cache를 의미한다.
  i. L2는 항상 L1의 super-set을 가진다. (즉, L1의 모든 블록들은 L2에도 있다)
  ii. L1과 L2 모두에서 miss 발생시 L1과 L2 모두로 블록을 로드 한다.
  iii. L1 miss, L2 hit 발생시 L1으로 해당 블록을 복사해 준다.
  iv. L2에서 블록이 evict될 때, 같은 블록이 L1에 존재할 경우 L1에서도 해당 블록을 evict한다.
- Write-back, write-allocate 방식을 사용한다. 이를 위해 더티 비트(dirty bit)를 구현한다.
- Output format중 clean eviction은 교체될 블록 (victim)이 clean 블록인 경우, dirty eviction은 교체될 블록이 dirty 블록인 경우이다.

B. Parameters

시뮬레이터는 L2의 용량(capacity), L2의 associativity, 블록 크기(block size)에 대한 parameter를 다음과 같은 범위에서 조정하여 실행할 수 있다.

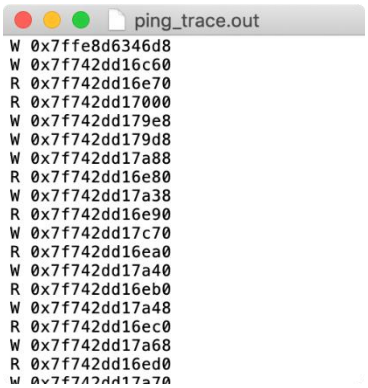
- Capacity: 4KB-1MB
- Associativity: 1-16
- Block size: 16B-128B

모든 parameter 값은 2의 거듭제곱이다.

L1 cache의 capacity와 associativity는 각각 L2 cache의 capacity와 associativity를 4로 나눈 값과 같다. L2의 associativity가 2이하인 경우 L1과 L2의 associativity가 같다고 가정한다. (i.e., 2일경우 L1, L2의 associativity가 동일하게 2가 된다). **L1과 L2의 block size는 동일하다.**

C. Trace File

SPEC 벤치마크의 동작을 기록한 트레이스 파일(trace file)은 읽기 혹은 쓰기 여부를 나타내기 위한 각각의 문자인 'R' 혹은 'W'와 16진수의 물리 주소(physical address)로 이루어져 있다. 시뮬레이터는 트레이스 파일을 입력으로 받고, 각 행의 물리 주소에 해당하는 캐시 블록에서 읽기 혹은 쓰기 작업을 수행한다.



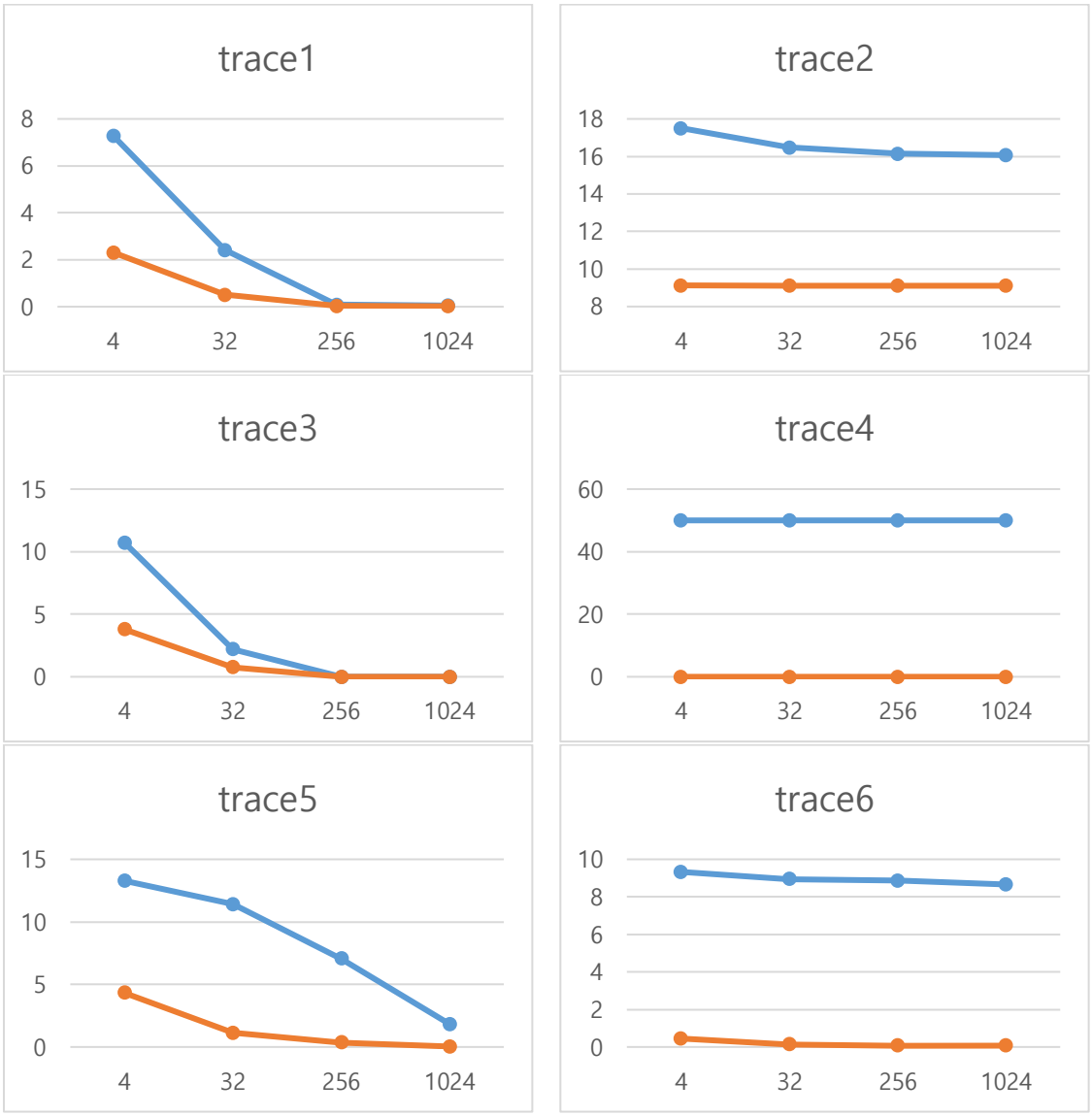
[Figure 1. An example of a trace file]

2. Cache Performance Characterization

구현한 시뮬레이터를 사용하여 주어진 트레이스 파일을 입력으로 하였을 때의 동작을 분석한다. 트레이스 파일의 종류, 전체 용량, 연관 정도, 블록 크기가 각각 달라지는 것에 따라 메모리 패턴이 변하는 것을 관찰할 수 있다. 분석 시, 가능한 모든 값의 조합을 실행할 필요는 없으며, 메모리 접근 특성을 설명하고 이해할 수 있는 조합을 선택하여 분석한다.

분석 시에는 반드시 그래프를 첨부한다. 예를 들어, 아래 그래프는 연관 정도를 4, 한 블록의 크기를 32B이고, LRU 블록 교체 정책을 사용하며, 캐시 용량을 4, 32, 256, 1024 KB

로 변경해가며 6개의 트레이스 파일을 실행하여 퍼센트 단위로 읽기 실패율(파란색)과 쓰기 실패율(주황색)을 기록한 결과이다. **아래와 같은 그래프를 각 블록 교체 정책 (LRU & Random) 에 대하여 그리고 비교 분석한다.**



3. Coding and Execution

A. Environment

제출된 코드는 Ubuntu 20.04, Windows Subsystem for Linux(Ubuntu 20.04) 환경에서 테스트될 것이다. 위 환경 중 하나에서 코드가 실행 가능하면 채점 가능하다. 이외의 OS 에서 코딩을 진행하는 경우 OS API 등의 코드로 인해 코드 컴파일이 불가능한 경우가 있을 수 있으니, 위 환경 중 최소 하나 이상에서 테스트한 후 제출하여야 한다.

B. Program Language

프로그래밍 언어는 C, C++만이 허용된다. 이 언어들 중에 어떤 것을 사용하여도 좋다.

- C 혹은 C++의 경우, gcc 4.8, g++ 4.8 버전 이상의 컴파일러를 이용하여 채점을 진행하게 된다.
- C 혹은 C++의 경우 Microsoft Visual Studio를 사용해야만 컴파일이 되는 경우, **컴파일이 불가능한 것으로 간주**한다. 또한, Microsoft Visual Studio 프로젝트 파일을 제출하는 것은 인정되지 않는다.

B. Execution Command

다음과 같은 명령어로 패러미터 및 트레이스 파일을 지정하여 실행한다.

```
$ ./runfile <-c capacity> <-a associativity> <-b block_size> <-lru 또는 -random> <trace file>
```

- `-c capacity`: L2 캐시의 전체 용량을 KB 단위의 capacity(4-1024)로 지정한다.
- `-a associativity`: L2 캐시의 연관 정도를 associativity(1-16)로 지정한다.
- `-b block_size`: 캐시의 블록 크기를 바이트 단위의 block_size(16-128)로 지정한다.
- `-lru`: Cache replacement 정책으로 LRU를 사용한다.
- `-random`: Cache replacement 정책으로 Random을 사용한다.
- 모든 parameter와 trace 파일은 프로그램을 실행할 때 반드시 필요하므로, 어느 것이라도 누락되는 경우에는 실행 오류로 처리한다.
- **L1 cache의 capacity와 associativity는 각각 L2 cache의 capacity와 associativity를 4로 나눈 값과 같다. L2의 associativity가 2이하인 경우 L1과 L2의 associativity가 같다고 가정**한다. (i.e., 2일경우 L1, L2의 associativity가 동일하게 2가 된다). **L1의 block size의 경우 L2의 block size와 동일하다.**

C. Input Format

입력 파일은 SPEC 벤치마크의 메모리 동작을 기록한 트레이스 파일(*.out)이다.

D. Output Format

시뮬레이터는 트레이스 파일의 마지막 내용까지 처리한 후 기록한 결과를 파일로 출력한 후 실행을 종료한다. 파일 내용의 예시는 아래와 같으며, 동일한 형식을 사용해야 한다 (아래의 값은 더미 값으로 실제 상황과는 다소 다를 수 있다). Miss rate의 경우 L1은 전체 접근 중 miss가 일어난 수의 비율을 표시하고, L2의 경우 L1 miss후

L2로 요청이 온 접근 중 miss가 일어난 수를 비율로 표시한다. (ex. L1 100번 접근 중 40번 miss 발생시 L1의 miss rate=40/100*100, L2는 L1에서 miss로 인해 발생된 40번의 접근 중에서 30번의 miss발생시 L2의 miss rate=30/40*100=75%이다.)

```
-- General Stats --
L1 Capacity:        4
L1 way:  4
L2 Capacity:        16
L2 way:             16
Block Size:         16
Total accesses: 200
Read accesses: 100
Write accesses: 100
L1 Read misses:40
L2 Read misses:30
L1 Write misses:40
L2 Write misses:30
L1 Read miss rate: 40%
L2 Read miss rate: 75%
L1 Write miss rate: 40%
L2 write miss rate: 75%
L1 Clean eviction: 20
L2 clean eviction: 30
L1 dirty eviction: 60
L2 dirty eviction: 60
```

출력 파일의 형식은 아래와 같다.

- workloadName_capacity_associativity_blockSize.out
- e.g. 400_perlbench_16_4_16.out

Report

과제를 수행한 학생들은 다음 내용이 들어있는 보고서를 작성하여 소스 파일과 함께 제출하여야 한다.

- 자신이 작성한 과제에 대한 간략한 설명
- Two Level Cache - 트레이스 파일 및 패러미터에 따른 결과 분석

- (C, C++) 과제의 컴파일 방법 및 컴파일 환경(사용한 OS, 컴파일러 버전).
- 과제의 실행 방법 및 실행 환경

결과 분석 시, **반드시 그래프 및 결과에 대한 설명을 첨부**한다. 그래프를 첨부하지 않거나, 설명(각 축이나 데이터의 값이 무엇을 의미하는지 등)이 미흡한 경우, **감점의 요인이 된다.**

컴파일 방법 및 컴파일 환경에 대한 보고서 작성 예시는 다음과 같다.

- ✓ WSL Ubuntu 20.04 환경에서 gcc 8.4.0 버전을 이용하여 다음 명령어로 컴파일 하였다.
  `gcc -o main main.c -std=c++14`
- ✓ Ubuntu 20.04 환경에서 g++ 8.4.0 버전을 이용하여 컴파일 하였다. 첨부한 makefile 을 소스 코드와 같은 폴더에 넣고, 리눅스 커맨드라인에서 `make` 명령어로 컴파일 할 수 있다.

실행 방법 및 실행 환경도 위와 같은 형식으로 작성한다. 단, 3-C Execution Command 에 지정한 명령어와 다른 명령어로 프로그램이 실행되는 경우, 감점의 요인이 될 수 있다. **3-B Programming Language에 명시 해놓은 gcc, g++ 이외의 컴파일러를 사용한 컴파일 방법을 보고서에 컴파일 방법으로 기재하였거나, 보고서에 컴파일 방법을 명시 해놓지 않은 경우 소스 코드를 컴파일 할 수 없는 것으로 간주한다.**

보고서는 ".pdf" 형식으로 변환하여 제출한다. 보고서 내에 소스 코드를 기재할 필요는 없다.

Submission

프로젝트 결과물(소스 파일) 및 보고서를 zip 형식으로 압축하여 LMS로 제출하도록 한다:

압축 파일의 이름은, 학번_이름.zip 으로 **꼭 양식을 지키도록** 한다. 대용량 첨부 파일의 유효 기간을 고려하여, **트레이스 파일의 경우에는 압축 파일에 첨부하지 않는다.** **지시 사항을 지키지 않을 경우, 감점의 요인이 된다.**

E. e.g. 202311999 김철수 학생의 경우, **202311999_김철수.zip** 형식으로 제출.

Notice

A. Due date: **23:59, 11th June, 2023**

B. Penalty

- Late due

Case	Penalty
1일 지연	점수 10% 감점
2일 지연	점수 30% 감점
3일 지연	점수 50% 감점
4일 이상 지연	0점 처리

- Cheating 적발 시 D+ 이하의 학점 부여
- 컴파일 혹은 실행이 마지막 Late due까지 안 될 경우 0점 처리.
- 파일 이름, 실행 방법 등이 규정한 양식과 다를 경우 감점 처리.
- 보고서에 컴파일 방법을 작성하지 않거나, 과제에서 규정한 컴파일러를 사용한 컴파일 방법이 적혀 있지 않을 경우, 컴파일이 불가능한 것으로 처리.

C. TA Contact

D. TA에게 질문 사항 또는 기타 용건이 있다면, 네 명의 TA 모두에게 질문 메일로 문의할 것(e.g. 수신 1명, 수신자 제외 참조 3명). TA를 직접 만나서 이야기하고 싶다면, 메일로 미리 약속을 잡을 것. 더불어 질문 메일의 제목 앞에는 항상 [컴퓨터구조 질문]을 붙인다.

A. **Minhyeok An(mh.an@dgist.ac.kr)**

B. **Jungbo Kim(jbkim@dgist.ac.kr)**

C. **Sungju Kim(sungju_kim@dgist.ac.kr)**

D. **YunHyeong Jeon(yhjeon@dgist.ac.kr)**