컴퓨터 구조 Homework #4

201911050 도담록, 2023-06-10

1. 과제 설명

Cache.h와 Cache.cpp에 Cache Class를 구현하여, Cache 내부에 접근 횟수와 miss 횟수, dirty/clean eviction count를 저장하도록 하였다. L1과 L2를 같은 클래스로 구현하였다

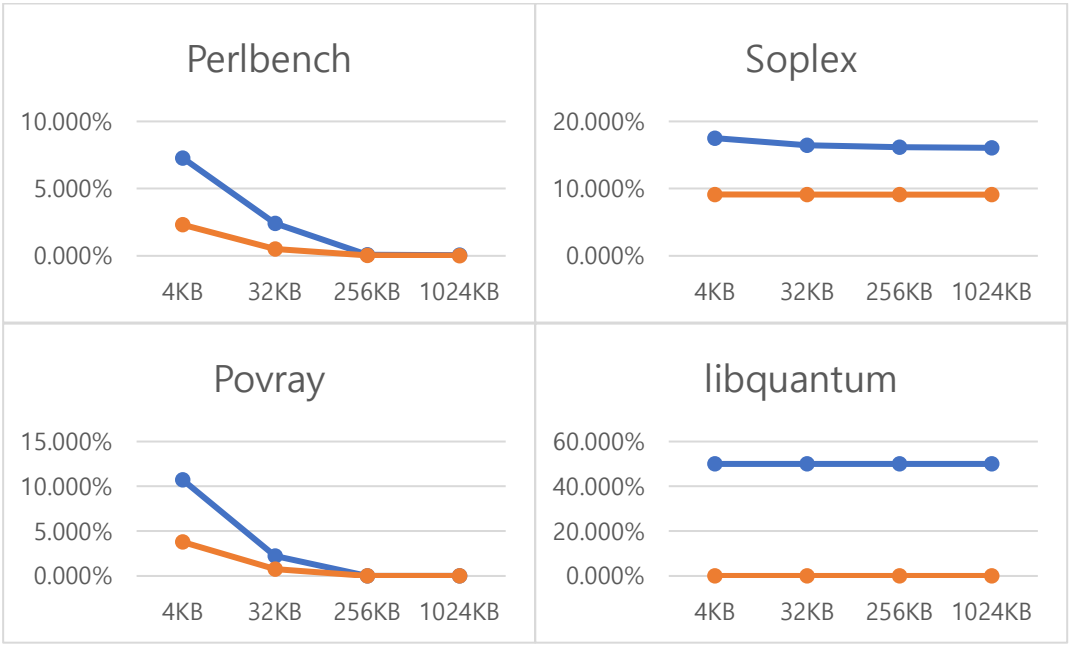
L1 cache에서 miss가 날 경우에 L2 cache에 같은 주소로 접근하도록 하였다.

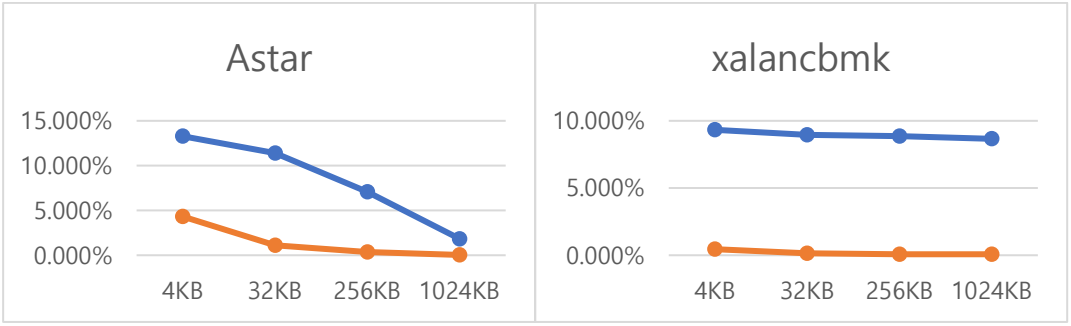
Input metadata를 받으면, capacity, associativity, block_size, policy를 통해 tag의 크기와 필요한 index의 수를 계산해, 두 개의 Cache를 만든다. Associativity, capacity는 L1 캐시가 L2 캐시의 1/4가 되도록 구현하였다. (Associativity가 1,2인 경우에는 같게) 프로그램을 실행하면, workloadname_capacity_associativity_block_size.out 파일이 과제 명세서에 적힌 대로 출력된다.

LRU의 경우에는 Set에 associativity size의 counter array를 넣어, 접근할 때마다 array의 모든 요소에 +1을 해주고, hit하거나 새로 데이터를 채워 넣을 때 0으로 초기화 해 주었다. Evict는 자동 evict와, 해당 tag를 evict하는 강제 evict 두 개를 구현하였다.

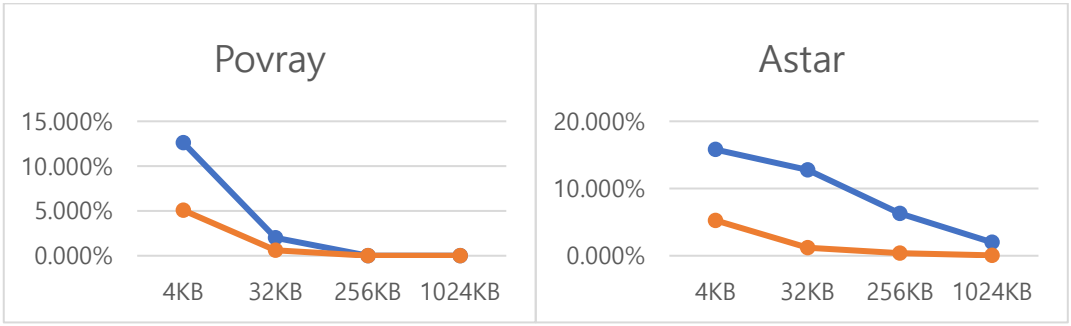
2. 결과 분석

1) Capacity c



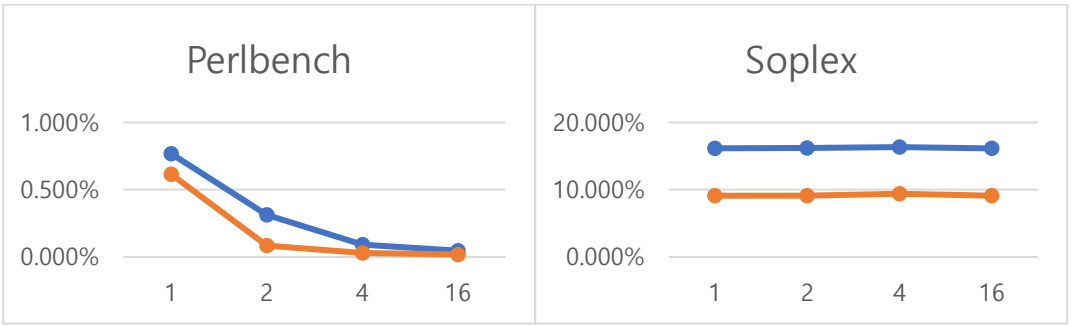


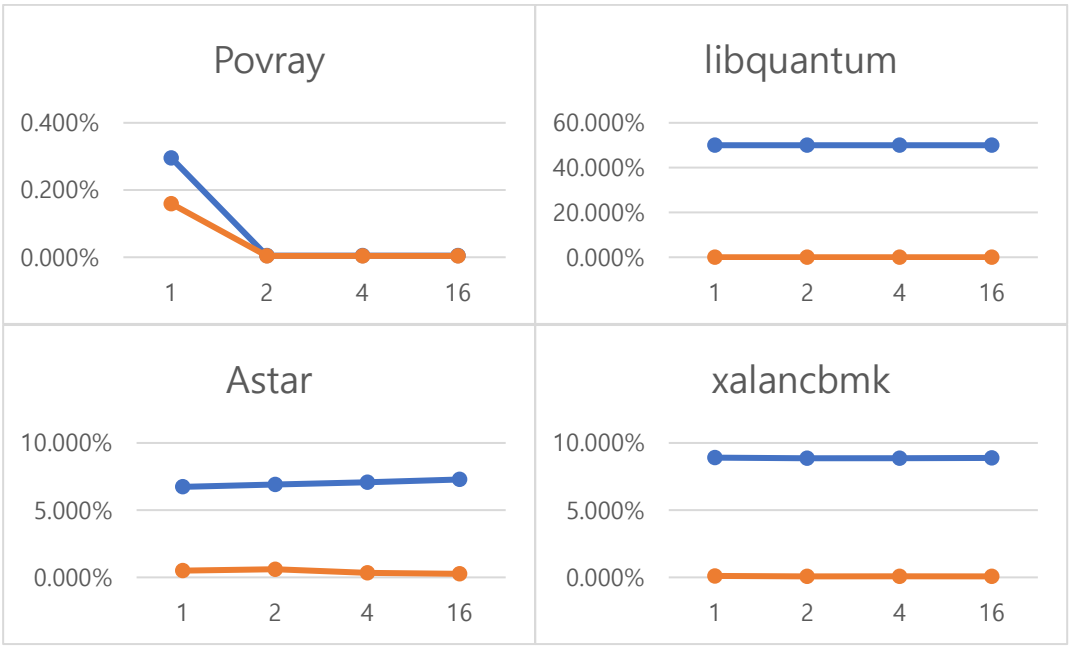
위의 그래프들은 LRU, associativity=4, block size = 32로 실행시킨 결과이다. 가로축은 KB 단위의 capacity, 세로축은 miss rate이다. 파란색은 read miss, 주황색은 write miss를 의미한다. (L2까지 miss인 rate). Capacity가 증가했기 때문에 대부분의 Benchmark들이 miss rate가 감소했지만, capacity보다는 block size, associativity가 중요한 Soplex, Libquantum, xalancbmk의 경우는 miss rate가 capacity와 무관해 보인다.



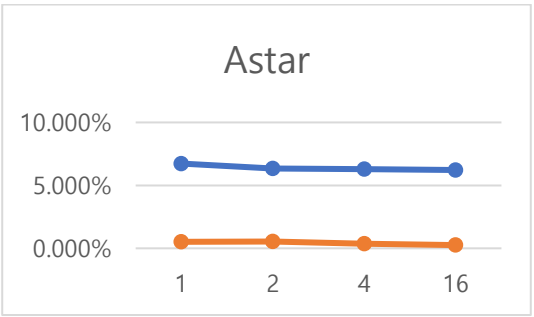
위의 그래프들은 같은 환경에서, LRU만 Random으로 바꾸어 준 그래프이다. 나머지 4개의 testbench에서는 유의미한 차이가 나지 않았으며, 이 두 testbench에서는 random이 LRU에 비해 miss rate가 높게 나왔다.

2) Associativity a



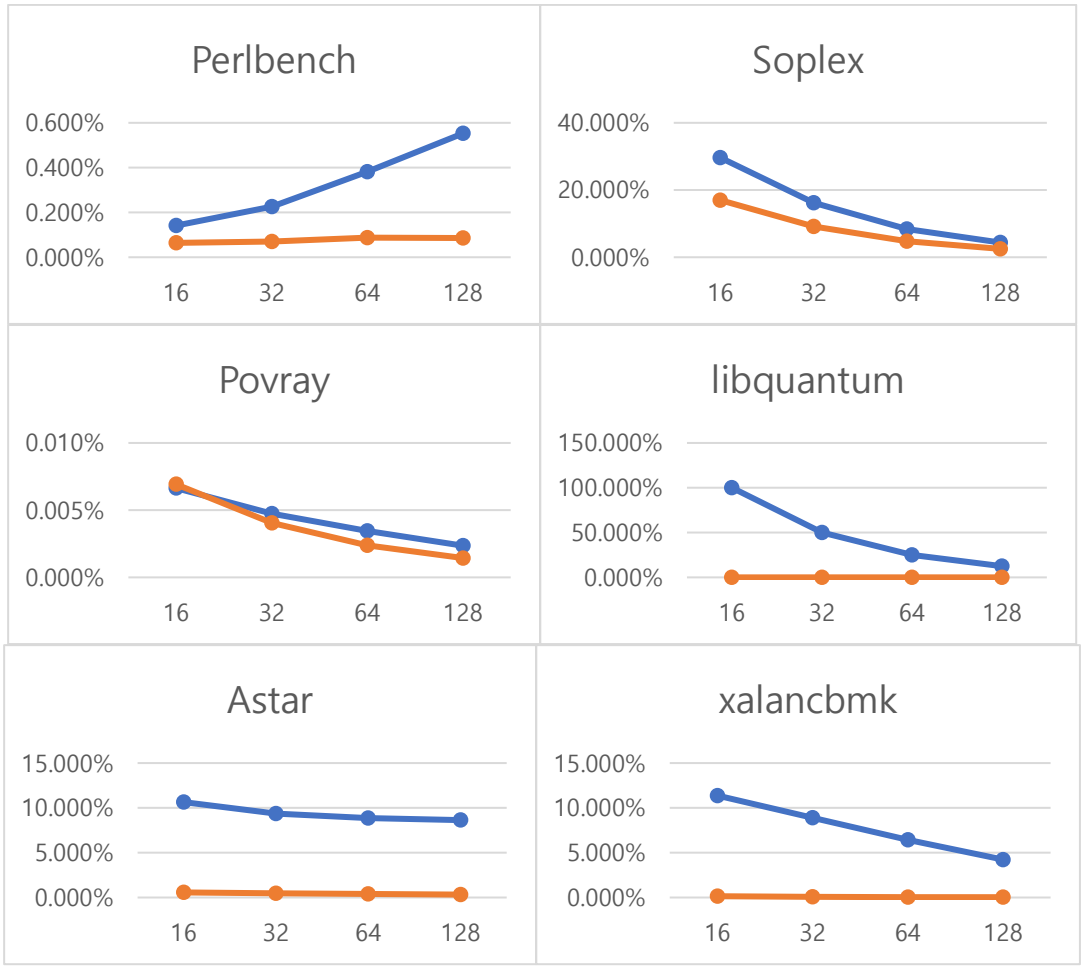


위 그래프들은 LRU, capacity=256, block size = 32로 실행시킨 결과이다. 가로축은 L2 associativity, 세로축은 miss rate이다. 파란색은 read miss, 주황색은 write miss를 의미한다. (L2까지 miss인 rate). Povray와 Perlbench의 경우 Associativity가 증가함에 따라 Miss rate가 줄어들었지만, 나머지 4개의 경우 Associativity가 miss rate에 영향을 미치지 않았다. 비슷한 주소에 의한 conflict miss가 잘 나지 않기 때문으로 보인다.



이 그래프는 같은 환경에서, Astar의 Evict 정책만 Random으로 바꾼 그래프이다. 나머지의 경우 Random보다는 LRU가 조금 더 Miss rate가 낮게 나왔지만, Astar의 경우에는 Random Associativty가 커짐에 따라 조금 더 좋게 나왔다. LRU는 오래 쓴 것을 다시 쓰지 않는다는 것을 가정하기 때문에, 데이터에 순환하면서 접근할 때에는 적합하지 않기 때문에, 특정한 program에 대해 비효율적으로 evict하여 random보다 성능이 떨어지는 것으로 보인다.

3) Block size b



위 그래프들은 LRU, capacity=256, associativity = 8로 실행시킨 결과이다. 가로축은 Byte 단위의 block size, 세로축은 miss rate이다. 파란색은 read miss, 주황색은 write miss를 의미한다. (L2까지 miss인 rate). Block size가 증가함에 따라(특히, Libquantum에서) Miss rate가 줄어드는 것을 확인할 수 있다. 하지만, Block size가 증가하면 일정 간격으로 분포된 데이터를 가져올 때 Miss rate를 키울 수 있다. Perlbench가 그 예시이다. 또한, Block size가 커질 경우 miss의 penalty가 커지기 때문에, block size가 커짐에 따라 Miss rate가 확연히 줄어들지 않는다면, 오히려 평균 접근 시간은 늘어날 수 있다.

위에서와 마찬가지로, Random policy를 사용할 경우 대부분의 경우 LRU에 비해 miss rate가 조금 증가하였으나, Astar의 경우에만 Miss rate가 조금 감소하는 모습을 보였다.

3. 과제의 컴파일 방법 및 컴파일 환경

WSL Ubuntu 20.04 환경에서 g++ 9.4.0 버전을 이용하여 컴파일 하였다. Cache.h, Cache.cpp, hw4.cpp, makefile을 같은 폴더에 넣고, 리눅스 커맨드에 make를 입력하면 컴파일되어 runfile이라는 실행 가능 파일이 나온다.

4. 과제의 실행 방법 및 실행 환경

컴파일해서 나온 파일인 runfile을 실행한다. 실행 또한 Ubuntu 20.04 환경에서 진행하였다.

과제 메뉴얼에 적힌 대로,

./runfile [-c capacity] [-a associativity] [-b block_size] [-lru or -random] <input file> 으로 실행한다. 모든 입력을 하지 않거나, 입력 범위를 벗어날 경우 오류 메시지를 출력하고, 실행되지 않도록 하였다.