2018년 2학기

컴퓨터 SW시스템 개론

CSED211

Lab Report #7

담당교수 : 김종

학번 : 20160074

학과 : 화학공학과

이름 : 고진민

POVIS ID : eric9709

--------------------------------------------------------------------------------------------------------------

<명예서약>

나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.

--------------------------------------------------------------------------------------------------------------

1. **구동 환경**

* Oracle VM VirtualBox 하에서 ubuntu로 구동, gcc 7.3.0 version

1. **Part A**

Part A는 trace 파일로부터 memory access data를 입력받아 hit/miss/eviction 여부를 판단하는 과제이다. 초기 입력으로 다양한 옵션을 입력받는다. 이러한 옵션의 입력 처리는 getopt 함수와 optarg, optind 변수를 사용해 처리한다(line 31~61). Parameter로 옵션의 종류를 인식한 뒤 switch case 문으로 세부 정보를 입력받아 처리한다.

옵션 별 입력 처리를 진행한 뒤에는 cache로 사용할 array를 malloc으로 동적할당한다. 이 때 data type은 사전에 지정한 cachetype이라는 struct이다(line 13 부근). 이 cachetype은 valid bit와 tag를 저장하는 struct이다. 옵션으로 입력받은 값들을 토대로 2차원 배열을 만들고, 이 때 row number는 block number이며 column number는 line number이다. 처음에는 valid 값을 전부 0으로 초기화한다.

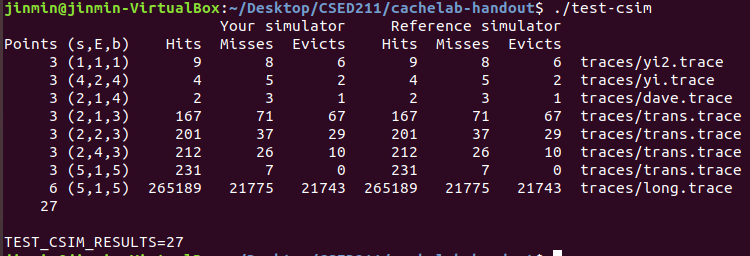
다음으로 trace 파일을 읽어 처리한다(line 75~). Wihle 루프를 사용해 fscanf를 이용해 line 단위로 읽으며 이 때 option에 data access type을, address에 주소를, size에 data size를 저장한다. 이 과정은 EOF를 입력받을 때까지 반복한다. 만약 data access type이 I인 경우면 바로 다음 루프를 돌린다. L 혹은 S인 경우는 cacheCheck 함수를 사용해 cache에서 어떤 일이 일어나는지 확인한다. hit/miss/miss eviction인 경우 각각에 대해 count를 1씩 상승시키고 다음 루프를 돌린다. 이 때 miss eviction은 miss count와 eviction count를 둘 다 1씩 올려야 한다. M인 경우는 L, S와 거의 같으나 마지막에 hit count를 1 추가해준다. 두 번째 경우는 무조건 hit이 뜨기 때문이다. 만약 처음에 -v 옵션이 사용되었다면 I인 경우를 제외하고 모두 결과를 출력해야 한다(line 125~161). 이 경우는 M인 경우와 S, L인 경우를 따로 분리해 경우마다 출력하도록 했다. 이 과정을 EOF가 될 때까지 반복한다.

마지막으로, 파일을 닫아주고 malloc으로 동적할당한 메모리를 free를 이용해 반환한다. 그리고 printSummary 함수를 호출하고, 프로그램을 종료한다.

사용자 지정 함수인 cacheCheck를 만들어 사용했다(line 176~). 이는 cache와 처음에 옵션으로 입력받은 s, E, b와 주소 address를 입력받아, hit인 경우는 0을, miss인 경우는 1을, miss eviction인 경우는 2를 반환하는 함수이다. 처음에는 주소의 크기를 계산한다. 원래 right shift로 구현했었으나 알 수 없는 문제가 생겨(아마도 부호비트 관련 문제일 것이다) 2로 계속 나눠 비트의 수를 세는 방법으로 진행했다. 이후 s를 이용해 set를 나타내는 bit를, b와 s를 이용해 tag를 나타내는 bit를 따로 추출해 낸다. 이들을 이용해 cache에서 hit 여부를 확인하고, miss인 경우는 LRU 알고리즘으로 cache에 데이터를 저장했다.

좀 더 구체적인 설명을 하자면, set을 찾아가 해당 set의 line을 전부 훑으며 valid bit가 valid하고 동시에 tag가 일치한다면 hitflag++를 한다. 이후 hit인 경우는 hit이 일어난 line의 데이터를 맨 앞(cache[setbit][0])으로 옮기고 다른 line은 하나씩 밀어준다. 이는 LRU 구현을 위함이다. Set 내에서 line number가 작을수록 최근에 사용된 cache이고, 클수록(뒤쪽일수록) 오래 전에 사용한 cache이다. 이 때문에 miss가 뜬 경우이면서 모든 cache가 valid한 경우(eviction)는 가장 뒤쪽의 cache를 삭제하고 하나씩 밀어준 뒤, 맨 앞에 새로운 cache를 저장하는 방법으로 LRU를 구현했다. 단순히 miss만 있는 경우는 하나씩 밀어주고 맨 앞에 새로운 cache를 넣는 방법으로 구현했다.

이 방법으로 test-csim을 실행하면 다음과 같은 결과를 얻을 수 있다.



1. **Part B**

Part B는 array transpose를 진행하는 함수 transpose\_submit를 구현하는 것이다. 기본적으로 세 경우 모두 blocking method를 이용해 구현을 진행했으나, N=64 인 경우는 단순히 blocking method만으로는 만점이 나오지 않아 miss를 줄이기 위해 직접 대입한 게 아닌, temp 변수를 이용해 대입하는 방법을 사용했다. 그러나 이 방법을 썼음에도 만점을 받을 수 있는 조건은 만족하지 못했다.

N=32인 경우는 block의 크기를 8로 설정해 blocking method를 적용했다. 4중 for 문으로 구현했고 처음의 2개는 block의 이동, 나중의 2개는 블록 내에서의 transpose 과정을 구현했다. 그 결과 큰 문제없이 만점 범위 내의 miss를 구할 수 있었다. 이 때 miss 수를 줄이기 위해서 diagonal case는 별도로 temp 변수를 이용해서 대입했다.

N=61인 경우 역시 N=32인 경우와 거의 같으나, block의 크기가 16이라는 점, 그리고 정사각행렬이 아니기 때문에 block 단위의 이동 후 블록 내부에서 작업을 진행할 때 실제 행렬을 벗어나지 않게 하는 장치가 주가로 필요하다는 점이 다르다. 이러한 방법으로 구현하면 역시 큰 문제없이 만점 범위 내의 miss를 구할 수 있었다.

N=64인 경우는 앞선 방법들로는 만점이 나오지 않아 miss를 줄이기 위해서 temp 변수를 이용해 대입하는 방법을 사용했다. 그러나 이 방법을 썼음에도 만점을 받을 수 있는 조건은 만족하지 못했다. 기본적으로는 block의 크기를 4로 했으며, 8로 늘린 경우는 오히려 miss가 더 많이 나왔다.

이 방법으로 driver.py를 실행하면 다음과 같은 결과를 얻을 수 있다.

