컴퓨터 구조 과제 4

컴퓨터 구조 3분반
소프트웨어학과 32170578 김산

Problem 1: Cache simulator

구현설명

Access state (line 13)
- 매크로를 이용해 load hit, load miss, store hit store miss를 정수로 표현했습니다.

Policy Map (line 20)
- Map을 통해 명령행 인자로 입력받은 policy 관련 문자열을 정수로 매핑 했습니다.
- wh_map (write hit policy)

Summary Information (line 25)
- 시뮬레이션의 결과를 저장하기 위한 전역변수를 선언하였습니다.

Cache configuration structure (line 34)
- 캐시의 설정값을 구조체로 정리하였습니다.

Cache class (line 44)
- 캐시를 클래스로 구현하였습니다.

- 생성자의 매개변수로 세트수, 라인수, 블록 바이트크기, write hit policy, write miss policy, replacement policy를 입력받도록 하였습니다.(line 62)

- cache line에 들어갈 valid, tag와 cache line의 참조 빈도를 저장하기 위한 frequency, cache line의 load 시간을 저장할 loadtime을 동적 할당하여 각 캐시 라인에 대한 정보를 저장 할 수 있도록 하였습니다.(line 75)

access_cache (line 87)
- 입력받은 주소를 set수. byte크기를 참고하여 tag, index, offset부분으로 나누어 저장하였습니다.
- input index를 통해 세트를 찾고 tag가 일치하고 valid가 1인 캐시라인을 찾고 있으면 cache hit로 함수를 반환합니다. 만약 없다면 cache miss로 처리합니다.
- write hit일 때 write through 방식이면 곧바로 메모리를 수정하기 때문에 메모리에 접근한다 생각하여 total cycle에 100을 더했고 write back 방식이면 쫓겨날 때만 메모리에 접근하기 때문에 캐시에 만 접근한다 생각하여 total cycle에 1을 더했습니다.
- write miss일 때 write-allocate 방식이면 메인 메모리의 블록을 수정해준 다음에 해당 블록을 캐시 메모리로 로드하여 캐시 라인을 갱신해주는 방식이기 때문에 캐시에 접근한다 생각하여 total cycle에 1을 더했고, 메인 메모리의 블록만 수정해주는 방식이므로 메모리에 접근한다 생각하여 total cycle에 100을 더하고 캐시를 갱신하는 작업을 하지 않고 함수를 반환하였습니다.

Replacement (line 133)
- replacement policy에 따라 lru, fifo, random 방식으로 replace를 수행합니다.

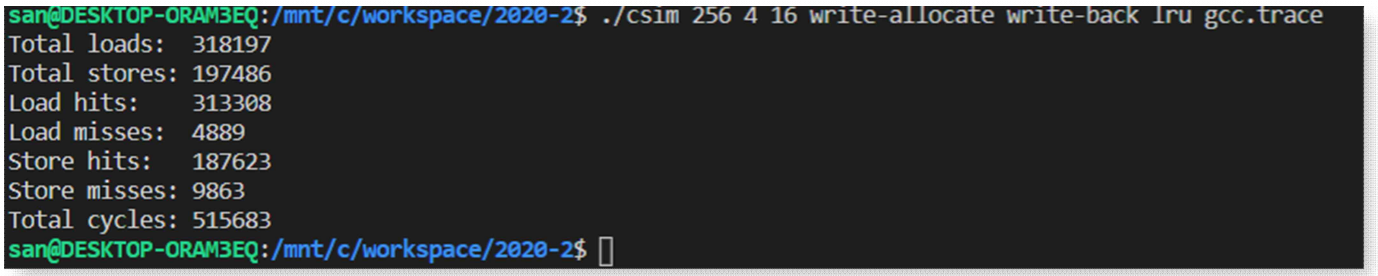
Set cache configuration (line 157)

- 명령행의 인자를 config 구조체에 정리하여 넣습니다. policy관련 문자열은 매핑하여 정수값으로 넣습니다.

Simulate (line 182)

- trace파일의 각 라인을 입출력 스트림을 통해 읽고 cache클래스의 access_cache함수를 통해 캐시를 시뮬레이션합니다.

수행결과



Problem 2: Best cache

"256 4 16 write-allocate write-back lru"를 기준으로
policy, set수, 세트당 line수, bytes를 바꿔 비교해 보았습니다.

Factor	Set			Line			byte		
	64	256	1024	1	4	16	1	4	16
Total load	318197	318197	318197	318197	318197	318197	318197	318197	318197
Total store	197486	197486	197486	197486	197486	197486	197486	197486	197486
Load hit	301674	313308	315523	298863	313308	315427	264152	303790	313308
Load miss	16523	4889	2674	19334	4889	2770	54045	14407	4889
Store hit	184895	187623	188403	185202	187623	188179	156174	166492	187623
Store miss	12591	9863	9083	12284	9863	9307	41312	30994	9863
Total cycle	515683	515683	515683	515683	515683	515683	515683	515683	515683
hit rate	94.02	97.06	97.67	93.47	97.06	97.60	77.31	90.35	97.06
miss rate	5.98	2.94	2.33	6.53	2.94	2.40	22.69	9.65	2.94
access time	6.98	3.94	3.33	7.53	3.94	3.40	23.69	10.65	3.94

	Write-through		Write-back	Replacement Policy		
	Write-allocate	No-write-allocate	Write-allocate	LRU	FIFO	random
Total load	318197	318197	318197	318197	318197	318197
Total store	197486	197486	197486	197486	197486	197486
Load hit	313308	310353	313308	313308	314171	313872
Load miss	4889	7844	4889	4889	4026	4325
Store hit	187623	164065	187623	187623	188047	187913
Store miss	9863	33421	9863	9863	9439	9573
Total cycle	515683	20066797	515683	515683	515683	515683
hit rate	97.06	91.30	97.06	97.06	97.32	97.23
miss rate	2.94	8.70	2.94	2.94	2.68	2.77
access time	3.94	9.70	3.94	3.94	3.68	3.77

결과를 통해 캐시의 크기가 커질수록 miss rate가 낮아져 access time이 줄어드는 것을 확인할 수 있었습니다. 이때 byte block의 크기에 따라 가장 큰 차이를 보였습니다. policy에서는 큰 차이는 없지만, Write-through와 No-write-allocate를 같이 사용하면 cycle 수가 크게 증가하고 성능이 크게 떨어짐을 확인 할 수 있었습니다.