학번 :
이름 : 김태훈

1. Part A: Writing a Cache Simulator

1-1. 기본적인 변수 선언, 자료형 선언, 구조체 선언

1-1-1. Cache_line 구조체 선언
캐시의 기본 구성요소인 라인 구조체를 만든다. 여기에는 valid bit, tag 를 저장하 는 tag 변수, 최소 최근 사용을 체크하기 위한 lru 변수가 있다. 실제로 이 캐시에 저장하지는 않으므로, block 은 만들지 않았다.

1-1-2. cache_set, cache 자료형 선언
캐시 라인이 E 개 모여서 캐시 set 이 되고, 캐시 set 이 S 개 모여서 캐시가 되므로, 다음과 같이 선언한다:
typedef Cache_line* Cache_set;
typedef Cache_set* Cache;

1-1-3. 다양한 변수 선언

int verbosity=0; // if verbosity=1 then print trace
int s=0; //set index bits
int b=0; //block offset bits
int E=0; //number of lines per set

char* trace_file = NULL; // trace file name

int S; // number of sets 2^s
int B; // block size 2^b

int miss_count=0;
int hit_count=0;
int eviction_count=0;
unsigned long long lru_counter = 1; // lru 가 높을 수록 최근에 사용한 것임

Cache cache; // 시뮬레이션에 사용할 캐시
mem_addr set_index_mask; // shift 로 block bit 를 없애고, tag + set bit 에서 set bit 만 추출

1-1-4. mem_addr 타입 선언
64 비트 기준이므로 memory address 의 타입을 unsigned long long 으로 선언한다.

1-2. main, replayTrace
csim-ref-skel.c 에서 가져왔다.

1-3. initCache()

cache 는 S 개의 Cache_set, Cache_set 은 E 개의 Cache_line 으로 구성되어있으므로
2 차원배열처럼 생각할 수 있다. 따라서 2 차원배열처럼 동적할당하고 초기화하면 된다.
또한 set_index_mask 는 block bit 를 제외한 tag + set bit 에서 set bit 만 추출해야하므로
set_index_mask = 00..0011..11 이어야 하고, 1 의 개수는 s 개여야 한다. 따라서
set_index_mask = (mem_addr)(pow(2,s)-1);

1-4. freeCache()

cache 를 2 차원배열처럼 생각하여 free 해주면 된다.

1-5. accessData(mem_addr addr)

먼저 메모리 주소 addr 에서 set_index 와 tag 를 추출한다. 각각 (addr>>b)&set_index_mask, addr>>(s+b)이다.

그리고 cache[set_index]에 있는 line 을 검사하면서 동일한 tag 가있는지 확인하고 있으면 hit count 를 올린다.
만약 없다면, miss 가 난 것이므로 빈 라인(valid==false)에 miss 가 난 데이터를 넣거나, 빈 라인이 없으면 lru 가 가장 작은 것을 방출하고 miss 가 난 데이터를 넣는다.

2. trans.c
test-trans.c 에 eval_perf(5,1,5)를 통해
s=5, E=1, b=5 임을 알 수 있다. 즉 set 는 32 개 , 라인은 1 개, 라인당 블록은 32 바이트(8 개의 int)임을 알 수 있다,
또한 driver.py 를 통해 테스트는 M 32 N 32 , M64 N64, M61 N67 임을 알 수 있다.

2-1. M 32 N 32

-각 표에서 matrix[i][j]에 대해 (tag,S(set index),B(block index))를 의미함
-각 한칸은 8*8

(0,0,0)	(0,0,28)	(0,1,0)	(0,1,28)	(0,2,0)	(0,2,28)	(0,3,0)	(0,3,28)
(0,28,0)	(0,28,28)	(0,29,0)	(0,29,28)	(0,30,0)	(0,30,28)	(0,31,0)	(0,31,28)
(1,0,0)	(1,0,28)	(1,1,0)	(1,1,28)	(1,2,0)	(1,2,28)	(1,3,0)	(1,3,28)
(1,28,0)	(1,28,28)	(1,29,0)	(1,29,28)	(1,30,0)	(1,30,28)	(1,31,0)	(1,31,28)
(2,0,0)	(2,0,28)	(2,1,0)	(2,1,28)	(2,2,0)	(2,2,28)	(2,3,0)	(2,3,28)
(2,28,0)	(2,28,28)	(2,29,0)	(2,29,28)	(2,30,0)	(2,30,28)	(2,31,0)	(2,31,28)
(3,0,0)	(3,0,28)	(3,1,0)	(3,1,28)	(3,2,0)	(3,2,28)	(3,3,0)	(3,3,28)

(3,28,0) (3,28,28)	(3,29,0) (3,29,28)	(3,30,0) (3,30,28)	(3,31,0) (3,31,28)

라인당 블록이 8 개의 int 를 저장할 수 있으므로 8*8 로 나누어 처리한다.
위의 표에서 알 수 있듯, i==j 일때 B[i][j] = A[j][i] 연산이 eviction 을 일으킨다. 따라서 그 부분만 따로 처리해주면 된다.

2-2 M=64 N=64

한칸=16*16

(0,0,0) (0,1,28) (3,24,0) (3,25,28)	(0,2,0) (0,3,28) (3,26,0) (3,27,28)	(0,4,0) (0,5,28) (3,28,0) (3,29,28)	(0,6,0) (0,7,28) (3,30,0) (3,31,28)
(4,0,0) (4,1,28) (7,24,0) (7,25,28)	(4,2,0) (4,3,28) (7,26,0) (7,27,28)	(4,4,0) (4,5,28) (7,28,0) (7,29,28)	(4,6,0) (4,7,28) (7,30,0) (7,31,28)
(8,0,0) (8,1,28) (11,24,0) (3,25,28)	(8,2,0) (8,3,28) (11,26,0) (11,27,28)	(8,4,0) (8,5,28) (11,28,0) (11,29,28)	(8,6,0) (8,7,28) (11,30,0) (11,31,28)
(12,0,0) (12,1,28) (15,24,0) (15,25,28)	(12,2,0) (12,3,28) (15,26,0) (15,27,28)	(12,4,0) (12,5,28) (15,28,0) (15,29,28)	(12,6,0) (12,7,28) (15,30,0) (15,31,28)

여기에는 겹치는 부분이 많다. 예를 들어 첫번째 칸에서 가로의 set bit 는 0 또는 1 인데, 세로의 set bit 가 0 인 것이 있어 (0,0,16) 와 (1,0,0) 이 겹친다.

정확히 알기 위해 맨 앞쪽 16*16 칸만 다시 보면

(0,0,0)	(0,0,4)	(0,0,8)	(0,0,12)	**(0,0,16)**	(0,0,20)	(0,0,24)	(0,0,28)
(0,8,0)	(0,8,4)	(0,8,8)	(0,8,12)	(0,8,16)	(0,8,20)	(0,8,24)	(0,8,28)
(0,16,0)	(0,16,4)	(0,16,8)	(0,16,12)	(0,16,16)	(0,16,20)	(0,16,24)	(0,16,28)
(0,24,0)	(0,24,4)	(0,24,8)	(0,24,12)	(0,24,16)	(0,24,20)	(0,24,24)	(0,24,28)
(1,0,0)	(1,0,4)	(1,0,8)	(1,0,12)	(1,0,16)	(1,0,20)	(1,0,24)	(1,0,28)
(1,8,0)	(1,8,4)	(1,8,8)	(1,8,12)	(1,8,16)	(1,8,20)	(1,8,24)	(1,8,28)
(1,16,0)	(1,16,4)	(1,16,8)	(1,16,12)	(1,16,16)	(1,16,20)	(1,16,24)	(1,16,28)

(1,24,0)	(1,24,4)	(1,24,8)	(1,24,12)	(1,24,16)	(1,24,20)	(1,24,24)	(1,24,28)

따라서 겹치는 것을 최소화 하기 위해 4*4 씩 잘라서 2-1 과같이 처리한다.

2-3. M61 N67

매트릭스를 8*8 로 나누어 2-1 과 같이 계산한다.