Multi Stream Ftl Simulator Report

2016311821 한승하

이번 과제에선 지난번 sector-based simulator코드의 많은 부분을 그대로 사용하였습니다. 따라서 지난 코드에서 수정부분 위주로 서술하였습니다.

<Global Variables>

```
#include "ftl.h"
#include "nand.h"
typedef struct l2ptable
{
        u32 ppn;
        char v;
}table;
table* l2p;
typedef struct ppntable
{
        char v;
        int s_id;
}ptable;
ptable *ppn;
u32 **ptr;
int *freeblocks;

#define MAX             0xFFFFFFFF
```

이번과제에서 사용한 Global variables입니다. 기본적으로 Sector based ftl과 동일한 global variables들을 사용하고 있으며, 각 변수들의 역할 또한 동일합니다. Ptr은 bank당 3개의 stream에 대한 ptr을 개별관리해주어야 하므로 2중 array로 수정해 주었습니다. 또한 ppn_table에 stream id를 기록할 수 있는 항목을 추가해 주었습니다.

<Ftl Open>

```
void ftl_open(void)
{
        freeblocks = (int*)malloc(sizeof(int)*N_BANKS);
        int j,k;
        for(k=0;k<N_BANKS;k++) freeblocks[k] = BLKS_PER_BANK;
        u32 freepages = N_BANKS*BLKS_PER_BANK*PAGES_PER_BLK;
        nand_init(N_BANKS,BLKS_PER_BANK,PAGES_PER_BLK);
        l2p = (table*)malloc(sizeof(table)*freepages);
        ppn = (ptable*)malloc(sizeof(ptable)*freepages);
        u32 i;
        for(i=0;i<freepages;i++)
        {
                l2p[i].v = 'n';
                ppn[i].v = 'f';
        }
        ptr = (u32**)malloc(sizeof(u32*)*N_BANKS);
        for(i=0;i<N_BANKS;i++) ptr[i] = (u32*)malloc(sizeof(u32)*3);
        for(i=0;i<N_BANKS;i++) for(j=0;j<3;j++) ptr[i][j] = j*PAGES_PER_BLK;
        for(i=0;i<N_BANKS;i++) for(j=0;j<3;j++)
        {
                ppn[i*BLKS_PER_BANK*PAGES_PER_BLK + ptr[i][j]].s_id = j;
        }
        return;
}
```

Ftl open함수입니다. 이전과제의 ftl open과 동일한 코드를 사용하고 있으며, ptr만 수정해 주었습

니다. S_id는 stream_id를 3으로 나눈 값으로 2중 array를 쉽게 관리하기 위해 관리해주었습니다.

또한 각 시작 ptr의 ppn_table에 s_id를 기록하여주었습니다.

<Ptr inc>

```
void inc_ptr(int bank, int s_id)
{
        u32 starting_addr = bank*BLKS_PER_BANK*PAGES_PER_BLK;
        if(ppn[starting_addr + ptr[bank][s_id]].v == 'f' || ppn[starting_addr + ptr[bank][s_id]].v == 'u') return;
        ptr[bank][s_id]++;
        while(1)
        {
                if(ptr[bank][s_id] >= BLKS_PER_BANK*PAGES_PER_BLK) ptr[bank][s_id] = 0;
                if(ppn[starting_addr + ptr[bank][s_id]].v == 'f') break;
                else if(ppn[starting_addr + ptr[bank][s_id]].v != 'f') ptr[bank][s_id] += PAGES_PER_BLK;
        }
        ppn[starting_addr + ptr[bank][s_id]].v = 'u';
        ppn[starting_addr + ptr[bank][s_id]].s_id = s_id;
}
```

Page_ptr을 증가시켜주기 위해 inc함수를 작성하였습니다. Page가 blk의 시작인 경우, 해당 blk가 사용된 blk인지 확인 후, 사용 중인 blk이라면 pages_per_blk만큼 증가시켜주며 free blk을 찾아서 return해 줍니다. 'u'는 사용을 위해 점 찍어 놓은 마크로 free에서 vaild가 되기 위한 준비상태에 있는 page입니다.

<Ftl Read>

```
void ftl_read(u32 lba, u32 num_sectors, u32 *read_buffer)
{
        u32 *buffer = (u32 *)calloc(SECTORS_PER_PAGE, sizeof(u32));
        int i;
        u32 read_buffer_ptr = 0;
        u32 num_sectors_to_read;
        u32 lpn = lba / SECTORS_PER_PAGE;
        u32 sect_offset = lba % SECTORS_PER_PAGE;
        u32 sectors_remain = num_sectors;
        while(sectors_remain != 0)
        {
                if(sect_offset + sectors_remain < SECTORS_PER_PAGE) num_sectors_to_read = sectors_remain;
                else num_sectors_to_read = SECTORS_PER_PAGE - sect_offset;
                lftl_read(lpn,buffer);
                for(i=0;i<num_sectors_to_read;i++)
                {
                        read_buffer[read_buffer_ptr + i] = buffer[sect_offset + i];
                }
                read_buffer_ptr += num_sectors_to_read;
                sect_offset = 0;
                sectors_remain -=num_sectors_to_read;
                lpn++;
                if(lpn == N_BANKS*BLKS_PER_BANK*PAGES_PER_BLK) lpn = 0;
        }
        return;
}
```

Ftl_Read입니다. Lba를 기준으로 하는 ftl read는 Sector-based ftl과 동일합니다.

Lpn이 최댓값을 넘어가는 경우를 고려해 if(lpn == MAX_LPN) lpn = 0; 부분을 수정해 주었습니다.

<Ftl Read by page>

```
void lftl_read(u32 lpn, u32 *read_buffer)
{
        if(ppn[l2p[lpn].ppn].v != 'v') return;
        int bank = lpn%N_BANKS;
        u32 addr = l2p[lpn].ppn;
        int blk = (addr - bank*BLKS_PER_BANK*PAGES_PER_BLK)/PAGES_PER_BLK;
        int page = (addr - bank*BLKS_PER_BANK*PAGES_PER_BLK - blk*PAGES_PER_BLK);
        u32 spare;
        nand_read(bank,blk,page,read_buffer,&spare);
        return;
}
```

Page 단위로 읽어주는 함수입니다. 만일 해당 page가 vaild상태가 아니라면 즉 읽으면 안되는 상황이라면 그냥 return해 줍니다.

<Ftl Write>

```
void ftl_write(u32 streamid, u32 lba, u32 num_sectors, u32 *write_buffer)
{
        u32 num_sectors_to_write;
        u32 *buffer = (u32 *)calloc(SECTORS_PER_PAGE, sizeof(u32));
        int i;
        u32 write_buffer_ptr = 0;
        u32 lpn = lba / SECTORS_PER_PAGE;
        u32 sect_offset = lba % SECTORS_PER_PAGE;
        u32 remain_sectors = num_sectors;
        while(remain_sectors != 0)
        {
                for(i=0;i<SECTORS_PER_PAGE;i++) buffer[i] = 0;
                if(sect_offset + remain_sectors >= SECTORS_PER_PAGE) num_sectors_to_write = SECTORS_PER_PAGE - sect_offset;
                else num_sectors_to_write = remain_sectors;
                if(sect_offset != 0 && l2p[lpn].v != 'n')
                {
                        lftl_read(lpn,buffer);
                }
                for(i=0;i<num_sectors_to_write;i++)
                {
                        buffer[sect_offset + i] = write_buffer[write_buffer_ptr + i];
                }
                lftl_write(lpn,buffer,streamid);
                s.ftl_write += SECTORS_PER_PAGE;
                write_buffer_ptr += num_sectors_to_write;
                sect_offset = 0;
                remain_sectors -= num_sectors_to_write;
                lpn++;
                if(lpn == N_BANKS*BLKS_PER_BANK*PAGES_PER_BLK) lpn = 0;
        }
        return;
}
```

Ftl_write함수입니다. Ftl_read와 마찬가지로 lba단위로 write 하는 코드는 sector-based ftl과 동일합니다.

Multi Stream ssd이므로 page_by_write 해주는 함수에 streamid도 같이 넘겨줍니다.

<Ftl Write by page>

```
void lftl_write(u32 lpn, u32 *write_buffer, int streamid)
{
        int gc_num = 0;
        int bank = lpn%N_BANKS;
        int s_id = streamid/3;
        int stream = streamid;
        inc_ptr(bank,s_id);
        if(ptr[bank][s_id]%PAGES_PER_BLK == 0) freeblocks[bank]--;
        while(freeblocks[bank] == 1)
        {
                gc_num = 1;
                garbage_collection(bank,stream);
                if(stream == 6) stream = 0;
                else stream += 3;
        }
        if(gc_num) inc_ptr(bank,s_id);
        if(l2p[lpn].v != 'n' && ppn[l2p[lpn].ppn].v == 'v')
        {
                u32 addr;
                addr = l2p[lpn].ppn;
                ppn[addr].v = 'i';
        }
        int blk = ptr[bank][s_id]/PAGES_PER_BLK;
        int page = ptr[bank][s_id]%PAGES_PER_BLK;
        l2p[lpn].ppn = bank*BLKS_PER_BANK*PAGES_PER_BLK + blk*PAGES_PER_BLK + page;
        l2p[lpn].v = 'w';
        nand_write(bank,blk,page,write_buffer,lpn);
        ppn[bank*BLKS_PER_BANK*PAGES_PER_BLK+ptr[bank][s_id]].v = 'v';
        return;
}
```

Ftl write by page입니다. 새로 쓸 page ptr을 inc함수를 통해 지정해준 뒤 만일 한 blk의 시작이라면 free block에서 1을 빼줍니다.

GC는 while문으로 구성되어 있는데 Stream 별로 invalid page가 충분하지 않아 새로운 free blk이 생기지 않는 경우를 위해서 위와 같이 구성하여 주었습니다. 만일 stream gc에서 free blk을 만들어주지 못했다면, 다른 stream을 gc하여 free blk을 만들어 줍니다.

이후 과정은 sector based ftl과 동일합니다.

<Garbage Collection>

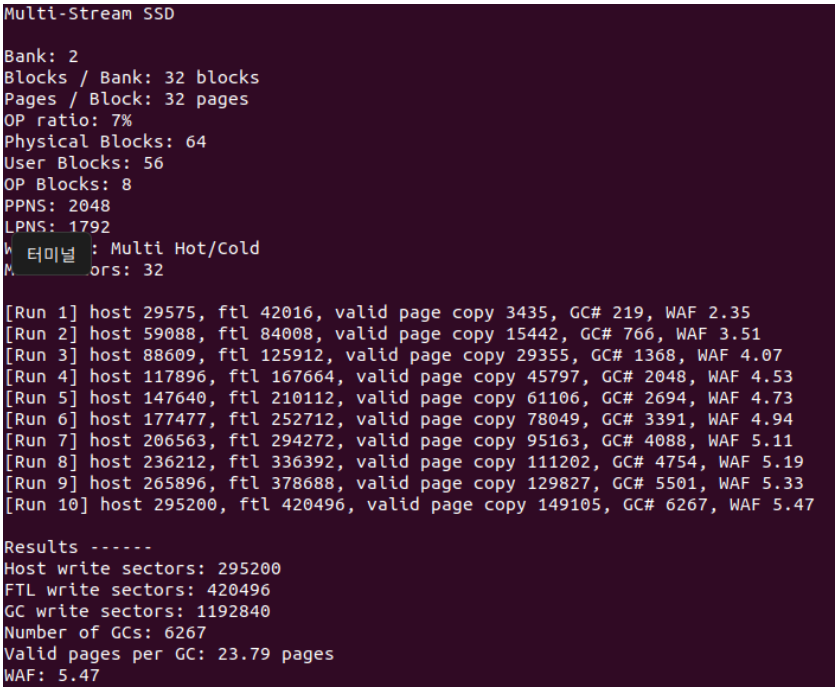
```
static void garbage_collection(u32 bank, u32 streamid)
{
        s.gc++;
        int i,j,count,st = 0;
        int s_id = streamid/3;
        int victim = 0;
        u32 spare;
        u32 r_buffer[SECTORS_PER_PAGE];
        memset(r_buffer, 0, DATA_SIZE);
        u32 s_point = bank*BLKS_PER_BANK*PAGES_PER_BLK;
        for(i=0;i<BLKS_PER_BANK;i++)
        {
                if(ppn[s_point+i*PAGES_PER_BLK].s_id != s_id) continue;
                count = 0;
                for(j=0;j<PAGES_PER_BLK;j++)
                {
                        if(ppn[s_point+i*PAGES_PER_BLK+j].v == 'i') count++;
                }
                if(count>=st)
                {
                        victim = i;
                        st = count;
                }
        }
        if(st == 0) return;
        for(j=0;j<PAGES_PER_BLK;j++)
        {
                if(ppn[s_point+victim*PAGES_PER_BLK+j].v == 'v')
                {
                        inc_ptr(bank,s_id);
                        nand_read(bank,victim,j,r_buffer,&spare);
                        nand_write(bank,ptr[bank][s_id]/PAGES_PER_BLK,ptr[bank][s_id]%PAGES_PER_BLK,r_buffer,spare);
                        ppn[bank*BLKS_PER_BANK*PAGES_PER_BLK + ptr[bank][s_id]].v = 'v';
                        ppn[bank*BLKS_PER_BANK*PAGES_PER_BLK + ptr[bank][s_id]].s_id = ppn[s_point+victim*PAGES_PER_BLK+j].s_id;
                        l2p[spare].ppn = bank*BLKS_PER_BANK*PAGES_PER_BLK + ptr[bank][s_id];
                        s.gc_write += SECTORS_PER_PAGE;
                }
        }
        for(j=0;j<PAGES_PER_BLK;j++) ppn[s_point+victim*PAGES_PER_BLK+j].v = 'f';
        nand_erase(bank,victim);
        freeblocks[bank]++;
        return;
}
```

Garbage collection 함수입니다. 이 또한 이전과제들과 victim을 선정하는 코드는 동일합니다.

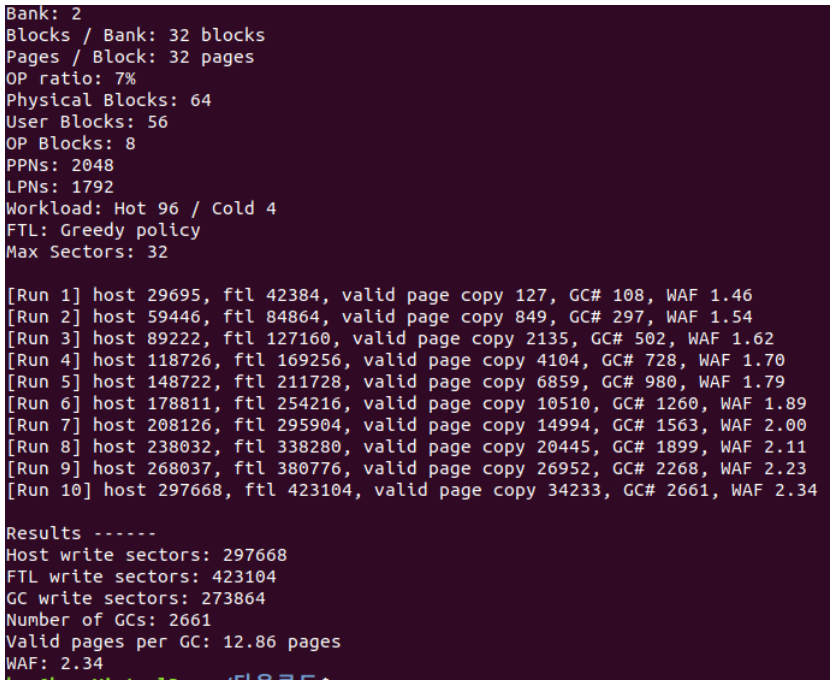
Victim이 선정되었다면 valid page를 모두 복사 후 erase 시켜줍니다. 이때 만일 invalid page가 없었다면 freeblocks를 증가시켜주지 않습니다. 이 경우 freeblocks가 그대로 이기 때문에 다음 stream에서 gc를 수행하게 됩니다.

만약 invilad page가 없는 경우 불필요한 동작을 줄이기 위해 바로 return하게 하였습니다.

<Result Analize>



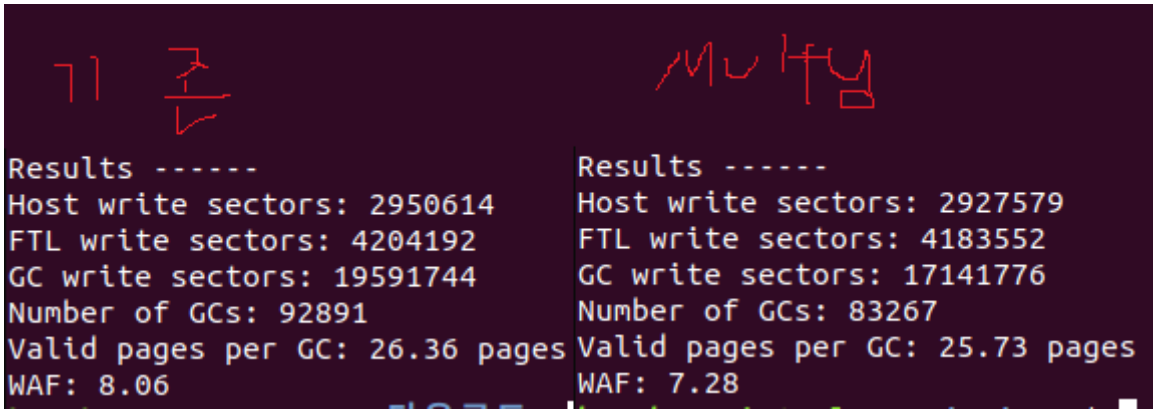
위는 이번 과제 Multi stream ssd의 result입니다.



위는 이전과제인 Sector based ftl을 같은 상황으로 돌렸을 경우입니다. 두 ftl을 비교했을 때 multi stream ssd이 약 2배 높은 waf 값을 보였습니다. 특히 GC 횟수가 눈에 띄게 증가했으며, 이는 한 번의 GC요청당 다수의 GC가 일어날 수 있기 때문이라고 분석하였습니다.

하지만 valid page per gc는 sector based ftl과 비교하여 약 2배 증가하였기에 multi stream ssd의 초기 목적은 달성된 것으로 보입니다. 하지만 실질적인 성능 향상은 어려운 것으로 보입니다. 그

러나 아래는 100run으로 simulation 했을 경우 multi stream ssd와 sector based ssd의 결과입니다.



놀랍게도 RUN 수가 늘어나자 오히려 Multi stream ssd가 더 좋은 성능을 보였습니다. Run 수가 늘어날수록 Hot/cold를 알고 따로 관리할 수 있다면, 시간이 지날수록 더 좋은 성능을 보일 것으로 보입니다.