Project 2 보고서

2조 한승하, 우제율

1.Code Description

1) Global Variables

```
typedef struct _ftl_statistics
{
    UINT32 gc_cnt;
    UINT32 page_wcount; // page write count
}ftl_statistics;
```

GC와 Page Write의 횟수를 세는데 사용되는 변수입니다.

```
typedef struct _misc_metadata
{
    UINT32 cur_write_vpn; // physical page for new write
    UINT32 cur_map_vpn; // phsical page for new map
    UINT32 cur_miscblk_vpn; // current write vpn for logging the misc. metadata
    UINT32 gc_vblock; // vblock number for garbage collection
    UINT32 gc_map_vblock; // vblock number for garbage collection
    UINT32 data_blk_cnt; // total number of data block count
    UINT32 map_blk_cnt; // total number of map block count
    UINT32 lpn_list_of_cur_vblock[PAGES_PER_BLK]; // logging lpn list of current write vblock for GC
    UINT32 gtd_list_of_cur_vblock[PAGES_PER_BLK]; // logging gtd list of current map vblock for Map GC
}misc_metadata; // per bank
```

전체적인 동작의 진행 상황을 저장하는 변수입니다.

"{UINT32} cur_write_vpn, cur_map_vpn, cur_miscblk_vpn"

= 앞으로 들어오는 Data(or Map)을 적어줄 VPN값을 저장하는 변수입니다. 현재 해당 VPN은 비어있습니다.

"{UINT32} gc_vblock, gc_map_block"

= 현재 Data(or Map) GC Block으로 정해진 Block의 Virtual Block Number를 저장하고 있습니다.

"{UINT32} data_blk_cnt, map_blk_cnt"

= 현재 사용하고 있는 Data(or Map) Block의 개수를 저장하고 있습니다. 최대 사용할 수 있는 개수를 초과하지 않도록 막고, GC를 실행하도록 하는데 사용됩니다.

"{UINT32} lpn_list_of_cur_vblock[PAGES_PER_BLK], gtd_list_of_cur_vblock[PAGES_PER_BLK]"

= cur_write_vpn(or cur_map_vpn)과 연계하여 사용되는 List입니다. 현재 사용중인 VBlock의 Page마다의 LPN(or GTD Index)를 가지고 있으며, Block 마지막 Page에 저장하는데 사용됩니다. GC를 할 때에 역추적하는 데에 요긴하게 사용됩니다.

```
static misc_metadata  g_misc_meta[NUM_BANKS];
static ftl_statistics g_ftl_statistics[NUM_BANKS];
static UINT32         g_bad_blk_count[NUM_BANKS];
```

"{static misc_metadata} g_misc_meta[NUM_BANKS]"

= 각 Bank별로 GC와 Page Write의 횟수를 세는데 사용되는 변수입니다.

"{static ftl_statistics} g_ftl_statistics[NUM_BANKS]"
= 각 Bank별로 전체적인 동작의 진행 상황을 저장하는 변수입니다.

"{static UINT32} g_bad_blk_count[NUM_BANKS]"
= 각 Bank별로 Bad Block의 개수를 저장하는 변수입니다.

```
UINT32                  g_ftl_read_buf_id;
UINT32                  g_ftl_write_buf_id;
```

"{UINT32} g_ftl_read_buf_id, g_ftl_write_buf_id"
= 각각 FTL Read와 FTL Write 동작에서 Buffer의 Index를 알려주는 변수입니다.

```
UINT32 bank_timer[NUM_BANKS] = { 0 };
UINT32 debug = 0;
```

"{UINT32} bank_timer[NUM_BANKS]"
= 각 Bank별로 Cache의 Age를 측정하고, 저장하기 위해서 사용된 변수입니다.

"{UINT32} debug"
= Debugging에 사용되는 변수입니다. If문을 이용하여 uart_printf를 출력할지 말지를 결정합니다.

2) Macros

```
#define VC_MAX              0xCDCD
#define MISCBLK_VBN         0x1 // vblock #1 <- misc metadata
#define META_BLKS_PER_BANK  (1 + 1) // include block #0, misc block
```

"VC_MAX" = 사용하는 않는 Block의 vcount값을 나타내는 Macro입니다.
"MISCBLK_VBN" = 어떤 Block을 MISC Block으로 사용하는지 알려주는 Macro입니다.
"META_BLKS_PER_BANK" = 0번째 Block과 MISC Block을 합친 즉, 접근하면 안돼는 Block의 개수를 알려주는 Macro입니다.

```
#define is_full_data_blks(bank) (g_misc_meta[bank].data_blk_cnt == 1)
#define inc_data_blk_cnt(bank)  (g_misc_meta[bank].data_blk_cnt--)
#define dec_data_blk_cnt(bank)  (g_misc_meta[bank].data_blk_cnt++)
#define is_full_map_blks(bank)  (g_misc_meta[bank].map_blk_cnt == 1)
#define inc_map_blk_cnt(bank)   (g_misc_meta[bank].map_blk_cnt--)
#define dec_map_blk_cnt(bank)   (g_misc_meta[bank].map_blk_cnt++)
#define inc_miscblk_vpn(bank)   (g_misc_meta[bank].cur_miscblk_vpn++)
```

"is_full_data_blks(bank), is_full_map_blks(bank)" = Data(or Map) Block의 개수를 전부 사용하였는지 확인하는 Macro입니다.
"inc_data_blk_cnt(bank), inc_map_blk_cnt(bank)" = Data(or Map) Block의 개수를 증가시키는 Macro입니다. 실제적으로는 남은 Block의 개수를 감소시켜 동작합니다.
"dec_data_blk_cnt(bank, dec_map_blk_cnt(bank)" = Data(or Map) Block의 개수를 감소시키는 Macro입니다. 실제적으로는 남은 Block의 개수를 증가시켜 동작합니다.

```
#define get_num_bank(lpn)              ((lpn) % NUM_BANKS)
#define get_bad_blk_cnt(bank)          (g_bad_blk_count[bank])
#define get_cur_write_vpn(bank)        (g_misc_meta[bank].cur_write_vpn)
#define set_new_write_vpn(bank, vpn)   (g_misc_meta[bank].cur_write_vpn = vpn)
#define get_cur_map_vpn(bank)          (g_misc_meta[bank].cur_map_vpn)
#define set_new_map_vpn(bank, vpn)     (g_misc_meta[bank].cur_map_vpn = vpn)
#define get_gc_vblock(bank)            (g_misc_meta[bank].gc_vblock)
#define set_gc_vblock(bank, vblock)    (g_misc_meta[bank].gc_vblock = vblock)
#define get_gc_map_vblock(bank)            (g_misc_meta[bank].gc_map_vblock)
#define set_gc_map_vblock(bank, vblock)    (g_misc_meta[bank].gc_map_vblock = vblock)
```

"get_num_bank(lpn)" = Bank Stripping하여 해당 LPN이 들어갈 Bank의 Number를 알려주는 Macro입니다.

"get_bad_blk_cnt(bank)" = Bad Block의 개수를 알려주는 Macro입니다.

"get_cur_write_vpn(bank), get_cur_map_vpn(bank)"

= 다음으로 저장할 Data(or Map) VPN의 값을 알려주는 Macro입니다.

"set_new_write_vpn(bank), set_new_map_vpn(bank)"

= 다음으로 저장할 Data(or Map) VPN의 값을 정하는 Macro입니다.

"get_gc_vblock(bank), get_gc_map_vblock(bank)"

= Data(or Map) GC Block의 Number를 알려주는 Macro입니다.

"set_gc_vblock(bank, vblock), set_gc_map_vblock(bank, vblock)"

= Data(or Map) GC Block의 Number를 정하는 Macro입니다.

```
#define set_lpn(bank, page_num, lpn)   (g_misc_meta[bank].lpn_list_of_cur_vblock[page_num] = lpn)
#define get_lpn(bank, page_num)        (g_misc_meta[bank].lpn_list_of_cur_vblock[page_num])
#define set_gtd(bank, page_num, gtd)   (g_misc_meta[bank].gtd_list_of_cur_vblock[page_num] = gtd)
#define get_gtd(bank, page_num)        (g_misc_meta[bank].gtd_list_of_cur_vblock[page_num])
#define get_miscblk_vpn(bank)          (g_misc_meta[bank].cur_miscblk_vpn)
#define set_miscblk_vpn(bank, vpn)     (g_misc_meta[bank].cur_miscblk_vpn = vpn)
#define CHECK_LPAGE(lpn)               ASSERT((lpn) < NUM_LPAGES)
#define CHECK_VPAGE(vpn)               ASSERT((vpn) < (VBLKS_PER_BANK * PAGES_PER_BLK))
```

"set_lpn(bank, page_num, lpn), set_gtd(bank, page_num, gtd)"

= 현재 사용중인 VBlock의 해당 Page가 어떤 LPN(or GTD Index)로 사용되고 있는지 저장하는 Macro입니다.

"get_lpn(bank, page_num), get_gtd(bank, page_num)"

= 현재 사용중인 VBlock의 해당 Page가 어떤 LPN(or GTD Index)로 사용되고 있는지 알려주는 Macro입니다.

"get_miscblk_vpn(bank), set_miscblk_vpn(bank, vpn)"

= 해당 Bank의 MISC Block의 VPN값을 알려주거나, 저장하는 Macro입니다.

"CHECK_LPAGE(lpn), CHECK_VPAGE(vpn)"

= 해당 LPN(or VPN)의 값이 Valid한 범위안에 있는지 확인하고 ASSERT해주는 Macro입니다.

3) Functions

```
static void   format(void);
static void   write_format_mark(void);
static void   sanity_check(void);
```

```
static void format(void)
{
    UINT32 bank, vblock, vcount_val;

    ASSERT(NUM_MISC_META_SECT > 0);
    ASSERT(NUM_VCOUNT_SECT > 0);

    uart_printf("Total FTL DRAM metadata size: %d KB", DRAM_BYTES_OTHER / 1024);

    uart_printf("VBLKS_PER_BANK: %d", VBLKS_PER_BANK);
    uart_printf("LBLKS_PER_BANK: %d", NUM_LPAGES / PAGES_PER_BLK / NUM_BANKS);
    uart_printf("META_BLKS_PER_BANK: %d", META_BLKS_PER_BANK);

    //----------------------------------------
    // initialize DRAM metadata
    //----------------------------------------
    mem_set_dram(VCOUNT_ADDR, NULL, VCOUNT_BYTES);
    mem_set_dram(VCOUNT_MAP_ADDR, NULL, VCOUNT_MAP_BYTES);
    mem_set_dram(CMT_ADDR, NULL, CMT_BYTES);
    mem_set_dram(GTD_ADDR, NULL, GTD_BYTES);
    mem_set_dram(PAGE_MAP_ADDR, NULL, PAGE_MAP_BYTES);
    mem_set_dram(MAP_BUF_ADDR, NULL, MAP_BUF_BYTES);

    //----------------------------------------
    // erase all blocks except vblock #0
    //----------------------------------------
    for (vblock = MISCBLK_VBN; vblock < VBLKS_PER_BANK; vblock++)
    {
        for (bank = 0; bank < NUM_BANKS; bank++)
        {
            vcount_val = VC_MAX;
            if (is_bad_block(bank, vblock) == FALSE)
            {
                nand_block_erase(bank, vblock);
                vcount_val = 0;
            }
            write_dram_16(VCOUNT_ADDR + ((bank * VBLKS_PER_BANK) + vblock) * sizeof(UINT16),
                          vcount_val);
            write_dram_16(VCOUNT_MAP_ADDR + ((bank * VBLKS_PER_BANK) + vblock) * sizeof(UINT16),
                          vcount_val);
        }
    }
    //----------------------------------------
    // initialize SRAM metadata
    //----------------------------------------
    init_metadata_sram();

    // flush metadata to NAND
    logging_misc_metadata();

    write_format_mark();
    led(1);
    uart_print("format complete");
}
```

"{static void} format(void)"

= "format"함수는 사용하는 DRAM의 사이즈와 Block들의 개수를 출력하는 동시에, 모든 Block들과 DRAM을 초기

화 시켜줍니다. 제가 DRAM에 추가한 영역 역시 초기화 시켜줍니다.

```
static void write_format_mark(void)
{
    // This function writes a format mark to a page at (bank #0, block #0).

    #ifdef __GNUC__
    extern UINT32 size_of_firmware_image;
    UINT32 firmware_image_pages = (((UINT32) (&size_of_firmware_image)) + BYTES_PER_FW_PAGE - 1) / BYTES_PER_FW_PAGE;
    #else
    extern UINT32 Image$$ER_CODE$$RO$$Length;
    extern UINT32 Image$$ER_RW$$RW$$Length;
    UINT32 firmware_image_bytes = ((UINT32) &Image$$ER_CODE$$RO$$Length) + ((UINT32) &Image$$ER_RW$$RW$$Length);
    UINT32 firmware_image_pages = (firmware_image_bytes + BYTES_PER_FW_PAGE - 1) / BYTES_PER_FW_PAGE;
    #endif

    UINT32 format_mark_page_offset = FW_PAGE_OFFSET + firmware_image_pages;

    mem_set_dram(FTL_BUF_ADDR, 0, BYTES_PER_SECTOR);

    SETREG(FCP_CMD, FC_COL_ROW_IN_PROG);
    SETREG(FCP_BANK, REAL_BANK(0));
    SETREG(FCP_OPTION, FO_E | FO_B_W_DRDY);
    SETREG(FCP_DMA_ADDR, FTL_BUF_ADDR);     // DRAM -> flash
    SETREG(FCP_DMA_CNT, BYTES_PER_SECTOR);
    SETREG(FCP_COL, 0);
    SETREG(FCP_ROW_L(0), format_mark_page_offset);
    SETREG(FCP_ROW_H(0), format_mark_page_offset);

    // At this point, we do not have to check Waiting Room status before issuing a command,
    // because we have waited for all the banks to become idle before returning from format().
    SETREG(FCP_ISSUE, NULL);

    // wait for the FC_COL_ROW_IN_PROG command to be accepted by bank #0
    while ((GETREG(WR_STAT) & 0x00000001) != 0);

    // wait until bank #0 finishes the write operation
    while (BSP_FSM(0) != BANK_IDLE);
}
```

"{static void} write_format_mark(void)"

= "write_format_mark"함수는 Firmware와 컴퓨터와의 통신에 사용되는 요소들을 설정하는 것으로 보입니다. Write 하는 동작의 형식을 정의하는 것으로 이해하였습니다.

```
static void sanity_check(void)
{
    UINT32 dram_requirement = RD_BUF_BYTES + WR_BUF_BYTES + COPY_BUF_BYTES + FTL_BUF_BYTES
        + HIL_BUF_BYTES + TEMP_BUF_BYTES + BAD_BLK_BMP_BYTES + VCOUNT_BYTES + VCOUNT_MAP_BYTES + CMT_BYTES + GTD_BYTES + PAGE_MAP_BYTES + MAP_BUF_BYTES;

    if ((dram_requirement > DRAM_SIZE) || // DRAM metadata size check
        (sizeof(misc_metadata) > BYTES_PER_PAGE)) // misc metadata size check
    {
        led_blink();
        while (1);
    }
}
```

"{static void} sanity_check(void)"

= "sanity_check"함수는 Board를 실제로 사용하기 전에 DRAM 사이즈의 요구사항이 적용 가능한지 판단해주는 함수로 이해하였습니다.

```
static void   load_misc_metadata(void);
static void   init_metadata_sram(void);
static void   load_metadata(void);
static void   logging_misc_metadata(void);
```

```c
static void load_misc_metadata(void)
{
    UINT32 misc_meta_bytes = NUM_MISC_META_SECT * BYTES_PER_SECTOR;
    UINT32 vcount_bytes    = NUM_VCOUNT_SECT * BYTES_PER_SECTOR;
    UINT32 vcount_addr     = VCOUNT_ADDR;
    UINT32 vcount_boundary = VCOUNT_ADDR + VCOUNT_BYTES;

    UINT32 load_flag = 0;
    UINT32 bank, page_num;
    UINT32 load_cnt = 0;

    flash_finish();

    disable_irq();
    flash_clear_irq();  // clear any flash interrupt flags that might have been set

    // scan valid metadata in descending order from last page offset
    for (page_num = PAGES_PER_BLK - 1; page_num != ((UINT32) -1); page_num--)
    {
        for (bank = 0; bank < NUM_BANKS; bank++)
        {
            if (load_flag & (0x1 << bank))
            {
                continue;
            }
            // read valid metadata from misc. metadata area
            nand_page_ptread(bank,
                             MISCBLK_VBN,
                             page_num,
                             0,
                             NUM_MISC_META_SECT + NUM_VCOUNT_SECT,
                             FTL_BUF(bank),
                             RETURN_ON_ISSUE);
        }
        flash_finish();

        for (bank = 0; bank < NUM_BANKS; bank++)
        {
            if (!(load_flag & (0x1 << bank)) && !(BSP_INTR(bank) & FIRQ_ALL_FF))
            {
                load_flag = load_flag | (0x1 << bank);
                load_cnt++;
            }
            CLR_BSP_INTR(bank, 0xFF);
        }
    }
```

```
    ASSERT(load_cnt == NUM_BANKS);

    for (bank = 0; bank < NUM_BANKS; bank++)
    {
        // misc. metadata
        mem_copy(&g_misc_meta[bank], FTL_BUF(bank), sizeof(misc_metadata));

        // vcount metadata
        if (vcount_addr <= vcount_boundary)
        {
            mem_copy(vcount_addr, FTL_BUF(bank) + misc_meta_bytes, vcount_bytes);
            vcount_addr += vcount_bytes;

        }
    }
    enable_irq();
}
```

"{static void} load_misc_metadata(void)"

= 모든 Metadata를 Load하여, Valid할 경우 vcount에 저장하여줍니다.

```c
static void init_metadata_sram(void)
{
    UINT32 bank;
    UINT32 vblock;

    //----------------------------------------
    // initialize misc. metadata
    //----------------------------------------
    for (bank = 0; bank < NUM_BANKS; bank++)
    {
        //TODO
        g_misc_meta[bank].data_blk_cnt  = VBLKS_PER_BANK - META_BLKS_PER_BANK - N_MAP_BLOCKS_PB;
        g_misc_meta[bank].data_blk_cnt -= get_bad_blk_cnt(bank);
        g_misc_meta[bank].map_blk_cnt   = N_MAP_BLOCKS_PB;

        // NOTE: vblock #0,1 don't use for user space
        write_dram_16(VCOUNT_ADDR + ((bank * VBLKS_PER_BANK) + 0) * sizeof(UINT16), VC_MAX);
        write_dram_16(VCOUNT_ADDR + ((bank * VBLKS_PER_BANK) + 1) * sizeof(UINT16), VC_MAX);
        write_dram_16(VCOUNT_MAP_ADDR + ((bank * VBLKS_PER_BANK) + 0) * sizeof(UINT16), VC_MAX);
        write_dram_16(VCOUNT_MAP_ADDR + ((bank * VBLKS_PER_BANK) + 1) * sizeof(UINT16), VC_MAX);

        //----------------------------------------
        // assign misc. block
        //----------------------------------------
        // assumption: vblock #1 = fixed location.
        // Thus if vblock #1 is a bad block, it should be allocate another block.
        set_miscblk_vpn(bank, MISCBLK_VBN * PAGES_PER_BLK - 1);
        ASSERT(is_bad_block(bank, MISCBLK_VBN) == FALSE);

        vblock = MISCBLK_VBN;
        //----------------------------------------
        // assign free block for gc
        //----------------------------------------
        do
        {
            vblock++;
            // NOTE: free block should not be secleted as a victim @ first GC
            write_dram_16(VCOUNT_ADDR + ((bank * VBLKS_PER_BANK) + vblock) * sizeof(UINT16), VC_MAX);
            write_dram_16(VCOUNT_MAP_ADDR + ((bank * VBLKS_PER_BANK) + vblock) * sizeof(UINT16), VC_MAX);
            // set free block
            set_gc_vblock(bank, vblock);

            ASSERT(vblock < VBLKS_PER_BANK);
        }while(is_bad_block(bank, vblock) == TRUE);
```

```
        //----------------------------------------
        // assign free block for map gc
        //----------------------------------------
        do
        {
            vblock++;
            // NOTE: free block should not be secleted as a victim @ first GC
            write_dram_16(VCOUNT_ADDR + ((bank * VBLKS_PER_BANK) + vblock) * sizeof(UINT16), VC_MAX);
            write_dram_16(VCOUNT_MAP_ADDR + ((bank * VBLKS_PER_BANK) + vblock) * sizeof(UINT16), VC_MAX);
            // set free block
            set_gc_map_vblock(bank, vblock);

            ASSERT(vblock < VBLKS_PER_BANK);
        }while(is_bad_block(bank, vblock) == TRUE);
        //----------------------------------------
        // assign free vpn for first new write
        //----------------------------------------
        do
        {
            vblock++;
            // 현재 next vblock부터 새로운 데이터를 저장을 시작
            set_new_write_vpn(bank, vblock * PAGES_PER_BLK);
            ASSERT(vblock < VBLKS_PER_BANK);
        }while(is_bad_block(bank, vblock) == TRUE);
        set_map_vcount(bank, vblock, VC_MAX);
        //----------------------------------------
        // assign free vpn for first new map
        //----------------------------------------
        do
        {
            vblock++;
            // 현재 next vblock부터 새로운 데이터를 저장을 시작
            set_new_map_vpn(bank, vblock * PAGES_PER_BLK);
            ASSERT(vblock < VBLKS_PER_BANK);
        }while(is_bad_block(bank, vblock) == TRUE);
        set_vcount(bank, vblock, VC_MAX);
        for (vblock = 0; vblock < VBLKS_PER_BANK; vblock++)
        {
            if (is_bad_block(bank, vblock) == TRUE)
                set_vcount(bank, vblock, VC_MAX);
        }
    }
}
```

"init_metadata_sram(void)"

= 모든 Bank들의 Block들에 대하여 vcount를 Initialize해줍니다. Metadata Block이거나 Bad Block일 경우, VC_MAX로 저장하여 접근을 막으며, 저는 Data Vcount와 Map Vcount를 나눠주어서 유동적으로 Block을 사용할 수 있도록 하였습니다.

```
static void load_metadata(void)
{
    load_misc_metadata();
}
```

"{static void} load_metadata(void)"

= "load_misc_metadata"함수를 호출합니다. 저희가 Power-Off Recovery를 구현하지 않아서 이 함수가 짧은 것이라고 생각이 됩니다.

```
static void logging_misc_metadata(void)
{
    UINT32 misc_meta_bytes = NUM_MISC_META_SECT * BYTES_PER_SECTOR; // per bank
    UINT32 vcount_addr     = VCOUNT_ADDR;
    UINT32 vcount_bytes    = NUM_VCOUNT_SECT * BYTES_PER_SECTOR; // per bank
    UINT32 vcount_boundary = VCOUNT_ADDR + VCOUNT_BYTES; // entire vcount data
    UINT32 bank;

    flash_finish();

    for (bank = 0; bank < NUM_BANKS; bank++)
    {
        inc_miscblk_vpn(bank);

        // note: if misc. meta block is full, just erase old block & write offset #0
        if ((get_miscblk_vpn(bank) / PAGES_PER_BLK) != MISCBLK_VBN)
        {
            nand_block_erase(bank, MISCBLK_VBN);
            set_miscblk_vpn(bank, MISCBLK_VBN * PAGES_PER_BLK); // vpn = 128
        }
        // copy misc. metadata to FTL buffer
        mem_copy(FTL_BUF(bank), &g_misc_meta[bank], misc_meta_bytes);

        // copy vcount metadata to FTL buffer
        if (vcount_addr <= vcount_boundary)
        {
            mem_copy(FTL_BUF(bank) + misc_meta_bytes, vcount_addr, vcount_bytes);
            vcount_addr += vcount_bytes;
        }
    }
    // logging the misc. metadata to nand flash
    for (bank = 0; bank < NUM_BANKS; bank++)
    {
        nand_page_ptprogram(bank,
                            get_miscblk_vpn(bank) / PAGES_PER_BLK,
                            get_miscblk_vpn(bank) % PAGES_PER_BLK,
                            0,
                            NUM_MISC_META_SECT + NUM_VCOUNT_SECT,
                            FTL_BUF(bank));
    }
    flash_finish();
}
```

"{static void} logging_misc_metadata(void)"

= Bank마다 Metadata Block을 지워준 뒤에, 새로운 Metadata를 그 자리에 적습니다.

```
static void write_page(UINT32 const lpn, UINT32 const sect_offset, UINT32 const num_sectors)
{
    CHECK_LPAGE(lpn);
    ASSERT(sect_offset < SECTORS_PER_PAGE);
    ASSERT(num_sectors > 0 && num_sectors <= SECTORS_PER_PAGE);

    UINT32 bank, old_vpn, new_vpn;
    UINT32 vblock, page_num, page_offset, column_cnt;

    bank        = get_num_bank(lpn); // page striping
    page_offset = sect_offset;
    column_cnt  = num_sectors;

    new_vpn  = assign_new_write_vpn(bank);
    old_vpn  = get_vpn(lpn);

    if (debug)
        uart_printf("write_page lpn = %u, bank = %u, old_vpn = %u, new_vpn = %u", lpn, bank, old_vpn, new_vpn);

    CHECK_VPAGE (old_vpn);
    CHECK_VPAGE (new_vpn);
    ASSERT(old_vpn != new_vpn);

    g_ftl_statistics[bank].page_wcount++;

    // if old data already exist,
    if (old_vpn != NULL)
    {
        extract_vpn(old_vpn, &vblock, &page_num);

        //--------------------------------------------------------------------------------------
        // `Partial programming'
        // we could not determine whether the new data is loaded in the SATA write buffer.
        // Thus, read the left/right hole sectors of a valid page and copy into the write buffer.
        // And then, program whole valid data
        //--------------------------------------------------------------------------------------
        if (num_sectors != SECTORS_PER_PAGE)
        {
            // Performance optimization (but, not proved)
            // To reduce flash memory access, valid hole copy into SATA write buffer after reading whole page
            // Thus, in this case, we need just one full page read + one or two mem_copy
            if ((num_sectors <= 8) && (page_offset != 0))
            {
                // one page async read
                nand_page_read(bank,
                               vblock,
                               page_num,
                               FTL_BUF(bank));
                // copy `left hole sectors' into SATA write buffer
```

```
            if (page_offset != 0)
            {
                mem_copy(WR_BUF_PTR(g_ftl_write_buf_id),
                         FTL_BUF(bank),
                         page_offset * BYTES_PER_SECTOR);
            }
            // copy `right hole sectors' into SATA write buffer
            if ((page_offset + column_cnt) < SECTORS_PER_PAGE)
            {
                UINT32 const rhole_base = (page_offset + column_cnt) * BYTES_PER_SECTOR;

                mem_copy(WR_BUF_PTR(g_ftl_write_buf_id) + rhole_base,
                         FTL_BUF(bank) + rhole_base,
                         BYTES_PER_PAGE - rhole_base);
            }
        }
        // left/right hole async read operation (two partial page read)
        else
        {
            // read `left hole sectors'
            if (page_offset != 0)
            {
                nand_page_ptread(bank,
                                 vblock,
                                 page_num,
                                 0,
                                 page_offset,
                                 WR_BUF_PTR(g_ftl_write_buf_id),
                                 RETURN_ON_ISSUE);
            }
            // read `right hole sectors'
            if ((page_offset + column_cnt) < SECTORS_PER_PAGE)
            {
                nand_page_ptread(bank,
                                 vblock,
                                 page_num,
                                 page_offset + column_cnt,
                                 SECTORS_PER_PAGE - (page_offset + column_cnt),
                                 WR_BUF_PTR(g_ftl_write_buf_id),
                                 RETURN_ON_ISSUE);
            }
        }
    }
    // full page write
    page_offset = 0;
    column_cnt  = SECTORS_PER_PAGE;
    // invalid old page (decrease vcount)
    set_vcount(bank, vblock, get_vcount(bank, vblock) - 1);
}
extract_vpn(new_vpn, &vblock, &page_num);
ASSERT(get_vcount(bank,vblock) < (PAGES_PER_BLK - 1));
```

"{static void} write_page(UINT32 const lpn, UINT32 const sect_offset, UINT32 const num_sectors)"

= FTL Write에 쓰이는 함수입니다. 해당 LPN이 이전에 할당된 Data Page가 있다면, Offset만큼 이동하여 새로운 Page에 적은 뒤 vcount를 감소시킵니다. 만약 이전에 할당된 Data Page가 없다면, 새로운 VPN을 받은 뒤 Write

하여준 뒤 vcount를 증가시킵니다. LPN List와 CMT를 갱신합니다.

```
static BOOL32 is_bad_block(UINT32 const bank, UINT32 const vblk_offset)
{
    if (tst_bit_dram(BAD_BLK_BMP_ADDR + bank*(VBLKS_PER_BANK/8 + 1), vblk_offset) == FALSE)
    {
        return FALSE;
    }
    return TRUE;
}
```

"{static BOOL32} is_bad_block(UINT32 const bank, UINT32 const vblk_offset)"

= 입력받은 "bank", "vblk_offset"의 VBlock이 Bad Block인지 판단하여 Bool값으로 반환합니다.

```
static BOOL32 check_format_mark(void)
{
    // This function reads a flash page from (bank #0, block #0) in order to check whether the SSD is formatted or not.

    #ifdef __GNUC__
    extern UINT32 size_of_firmware_image;
    UINT32 firmware_image_pages = (((UINT32) (&size_of_firmware_image)) + BYTES_PER_FW_PAGE - 1) / BYTES_PER_FW_PAGE;
    #else
    extern UINT32 Image$$ER_CODE$$RO$$Length;
    extern UINT32 Image$$ER_RW$$RW$$Length;
    UINT32 firmware_image_bytes = ((UINT32) &Image$$ER_CODE$$RO$$Length) + ((UINT32) &Image$$ER_RW$$RW$$Length);
    UINT32 firmware_image_pages = (firmware_image_bytes + BYTES_PER_FW_PAGE - 1) / BYTES_PER_FW_PAGE;
    #endif

    UINT32 format_mark_page_offset = FW_PAGE_OFFSET + firmware_image_pages;
    UINT32 temp;

    flash_clear_irq();  // clear any flash interrupt flags that might have been set

    SETREG(FCP_CMD, FC_COL_ROW_READ_OUT);
    SETREG(FCP_BANK, REAL_BANK(0));
    SETREG(FCP_OPTION, FO_E);
    SETREG(FCP_DMA_ADDR, FTL_BUF_ADDR);     // flash -> DRAM
    SETREG(FCP_DMA_CNT, BYTES_PER_SECTOR);
    SETREG(FCP_COL, 0);
    SETREG(FCP_ROW_L(0), format_mark_page_offset);
    SETREG(FCP_ROW_H(0), format_mark_page_offset);

    // At this point, we do not have to check Waiting Room status before issuing a command,
    // because scan list loading has been completed just before this function is called.
    SETREG(FCP_ISSUE, NULL);

    // wait for the FC_COL_ROW_READ_OUT command to be accepted by bank #0
    while ((GETREG(WR_STAT) & 0x00000001) != 0);

    // wait until bank #0 finishes the read operation
    while (BSP_FSM(0) != BANK_IDLE);

    // Now that the read operation is complete, we can check interrupt flags.
    temp = BSP_INTR(0) & FIRQ_ALL_FF;

    // clear interrupt flags
    CLR_BSP_INTR(0, 0xFF);

    if (temp != 0)
    {
        return FALSE;   // the page contains all-0xFF (the format mark does not exist.)
    }
    else
    {
        return TRUE;    // the page contains something other than 0xFF (it must be the format mark)
    }
}
```

"{static BOOL32} check_format_mark(void)"

= "load_format_mark"와는 코드는 매우 비슷하지만, "temp"를 통하여서 Format Mark를 제대로 가지고 있는지 체크합니다. 즉, 제대로 형식을 갖추고 있는지 체크하는 함수인 것 같습니다.

```
void ftl_open(void)
{
    // debugging example 1 - use breakpoint statement!
    /* *(UINT32*)0xFFFFFFFE = 10; */

    /* UINT32 volatile g_break = 0; */
    /* while (g_break == 0); */

    led(0);
    sanity_check();
    //----------------------------------------
    // read scan lists from NAND flash
    // and build bitmap of bad blocks
    //----------------------------------------
    build_bad_blk_list();

    //----------------------------------------
    // If necessary, do low-level format
    // format() should be called after loading scan lists, because format() calls is_bad_block().
    //----------------------------------------
/*  if (check_format_mark() == FALSE) */


    if (TRUE)
    {
        uart_print("do format");
        format();
        uart_print("end format");
    }
    // load FTL metadata
    else
    {
        load_metadata();
    }
    g_ftl_read_buf_id = 0;
    g_ftl_write_buf_id = 0;

    // This example FTL can handle runtime bad block interrupts and read fail (uncorrectable bit errors) interrupts
    flash_clear_irq();

    SETREG(INTR_MASK, FIRQ_DATA_CORRUPT | FIRQ_BADBLK_L | FIRQ_BADBLK_H);
    SETREG(FCONF_PAUSE, FIRQ_DATA_CORRUPT | FIRQ_BADBLK_L | FIRQ_BADBLK_H);

    enable_irq();
}
```

"{void} ftl_open(void)"

= 전체적인 FTL Initialize를 해주는 함수입니다. 역할은 FTL Simulation때 사용한 "ftl_open"과 비슷하다고 생각됩니다.

```
void ftl_read(UINT32 const lba, UINT32 const num_sectors);
```

"{void} ftl_read(UINT32 const lba, UINT32 const num_sectors)"

= 기존의 코드를 거의 수정하지 않았습니다. While문을 이용하여 입력받은 만큼의 Sector를 모두 써줍니다. 차이가 있다면 "get_vpn"함수를 이용하여 CMT를 갱신하는 코드를 넣어주었습니다.

```
bank = get_num_bank(lpn); // page striping
vpn  = get_vpn(lpn);
CHECK_VPAGE(vpn);
extract_vpn(vpn, &vpn_blk, &vpn_page);

if (vpn != NULL)
{
    nand_page_ptread_to_host(bank,
                             vpn_blk,
                             vpn_page,
                             sect_offset,
                             num_sectors_to_read);
}
```

```
void ftl_write(UINT32 const lba, UINT32 const num_sectors);
```

"{void} ftl_write(UINT32 const lba, UINT32 const num_sectors)"

= 기존의 코드를 그대로 사용하였습니다. 실질적으로 대부분의 동작은 "write_page"에서 하기 때문에 이 함수에서는 앞으로 남은 Sector에 대한 관리를 해주기만 합니다.

```
void ftl_test_write(UINT32 const lba, UINT32 const num_sectors)
{
    ASSERT(lba + num_sectors <= NUM_LSECTORS);
    ASSERT(num_sectors > 0);

    ftl_write(lba, num_sectors);
}
```

"{void} ftl_test_write(UINT32 const lba, UINT32 const num_sectors)"

= FTL Test에 사용되는 함수라고 생각이 됩니다.

```
void ftl_flush(void)
{
    /* ptimer_start(); */
    logging_misc_metadata();
    /* ptimer_stop_and_uart_print(); */
}
```

"{void} ftl_flush(void)"

= Metadata를 전부 Write하는데에 사용되는 함수입니다.

```
void ftl_test_write(UINT32 const lba, UINT32 const num_sectors);
void ftl_flush(void);
void ftl_isr(void);
```

```
void ftl_isr(void)
{
    UINT32 bank;
    UINT32 bsp_intr_flag;

    uart_print("BSP interrupt occured...");
    // interrupt pending clear (ICU)
    SETREG(APB_INT_STS, INTR_FLASH);

    for (bank = 0; bank < NUM_BANKS; bank++) {
        while (BSP_FSM(bank) != BANK_IDLE);
        // get interrupt flag from BSP
        bsp_intr_flag = BSP_INTR(bank);

        if (bsp_intr_flag == 0) {
            continue;
        }
        UINT32 fc = GETREG(BSP_CMD(bank));
        // BSP clear
        CLR_BSP_INTR(bank, bsp_intr_flag);

        // interrupt handling
        if (bsp_intr_flag & FIRQ_DATA_CORRUPT) {
            uart_printf("BSP interrupt at bank: 0x%x", bank);
            uart_print("FIRQ_DATA_CORRUPT occured...");
        }
        if (bsp_intr_flag & (FIRQ_BADBLK_H | FIRQ_BADBLK_L)) {
            uart_printf("BSP interrupt at bank: 0x%x", bank);
            if (fc == FC_COL_ROW_IN_PROG || fc == FC_IN_PROG || fc == FC_PROG) {
                uart_print("find runtime bad block when block program...");
            }
            else {
                uart_printf("find runtime bad block when block erase...vblock #: %d", GETREG(BSP_ROW_H(bank)) / PAGES_PER_BLK);
                ASSERT(fc == FC_ERASE);
            }
        }
    }
}
```

"{void} ftl_isr(void)"

= BSP Interrupt가 발행하면, 어느 Bank에서 어느 현상이 발생하였는지 친절하게 출력하여주는 함수입니다.

4) My Own Functions

```
static UINT32 get_entry_num(UINT32 const lpn)
    { return (lpn / NUM_BANKS) % N_MAP_ENTRIES_PER_PAGE; }
static UINT32 get_gtd_idx(UINT32 const lpn)
    { return (lpn / NUM_BANKS) / N_MAP_ENTRIES_PER_PAGE; }
static UINT32 cmt_entry_addr(UINT32 const bank, UINT32 const cache_slot, UINT32 const entry_num)
    { return CMT_ADDR + (((bank * N_CACHED_MAP_PAGE_PB) + cache_slot) * CMT_ROW_BYTES) + (entry_num * sizeof(UINT32)); }
static UINT32 cmt_lpn_addr(UINT32 const bank, UINT32 const cache_slot)
    { return CMT_ADDR + (((bank * N_CACHED_MAP_PAGE_PB) + cache_slot + 1) * CMT_ROW_BYTES) - (sizeof(UINT32) * 3); }
static UINT32 cmt_dirty_addr(UINT32 const bank, UINT32 const cache_slot)
    { return CMT_ADDR + (((bank * N_CACHED_MAP_PAGE_PB) + cache_slot + 1) * CMT_ROW_BYTES) - (sizeof(UINT32) * 2); }
static UINT32 cmt_time_addr(UINT32 const bank, UINT32 const cache_slot)
    { return CMT_ADDR + (((bank * N_CACHED_MAP_PAGE_PB) + cache_slot + 1) * CMT_ROW_BYTES) - (sizeof(UINT32) * 1); }
static UINT32 gtd_idx_addr(UINT32 const bank, UINT32 const gtd_idx)
    { return GTD_ADDR + (bank * GTD_ROW_BYTES) + (gtd_idx * sizeof(UINT32)); }
```

"{static UINT32} get_entry_num(UINT32 const lpn)"

= 입력받은 "lpn"으로부터 Cache의 Entry Number를 계산하는 함수입니다.

"{static UINT32} get_gtd_idx(UINT32 const lpn)"

= 입력받은 "lpn"으로부터 GTD의 Index를 계산하는 함수입니다.

"{static UINT32} cmt_entry_addr(UINT32 const bank, UINT32 const cache_slot, UINT32 const entry_num)"

= 입력받은 "bank", "cache_slot", "entry_num"의 VPN이 저장되어 있는 DRAM Address를 계산하는 함수입니다.

"{static UINT32} cmt_lpn_addr(UINT32 const bank, UINT32 const cache_slot)"

= 입력받은 "bank", "cache_slot"의 LPN이 저장되어 있는 DRAM Address를 계산하는 함수입니다.

"{static UINT32} cmt_dirty_addr(UINT32 const bank, UINT32 const cache_slot)"
= 입력받은 "bank", "cache_slot"의 Dirty Bit가 저장되어 있는 DRAM Address를 계산하는 함수입니다.

"{static UINT32} cmt_time_addr(UINT32 const bank, UINT32 const cache_slot)"
= 입력받은 "bank", "cache_slot"의 Time(or Age)가 저장되어 있는 DRAM Address를 계산하는 함수입니다.

"{static UINT32} gtd_idx_addr(UINT32 const bank, UINT32 const gtd_idx)"
= 입력받은 "bank", gtd_idx"의 VPN이 저장되어 있는 DRAM Address를 계산하는 함수입니다.

```
static void   garbage_collection(UINT32 const bank);
static void   garbage_collection_map(UINT32 const bank);
```

```
static void garbage_collection(UINT32 const bank)
{
    //uart_printf("Data GC bank = %u", bank);
    ASSERT(bank < NUM_BANKS);
    g_ftl_statistics[bank].gc_cnt++;

    UINT32 src_lpn;
    UINT32 vt_vblock;
    UINT32 free_vpn;
    UINT32 vcount; // valid page count in victim block
    UINT32 src_page;
    UINT32 gc_vblock;

    g_ftl_statistics[bank].gc_cnt++;

    vt_vblock = get_vt_vblock(bank);   // get victim block
    vcount    = get_vcount(bank, vt_vblock);
    gc_vblock = get_gc_vblock(bank);
    free_vpn  = gc_vblock * PAGES_PER_BLK;

/*     uart_printf("garbage_collection bank %d, vblock %d",bank, vt_vblock); */

    ASSERT(vt_vblock != gc_vblock);
    ASSERT(vt_vblock >= META_BLKS_PER_BANK && vt_vblock < VBLKS_PER_BANK);
    ASSERT(vcount < (PAGES_PER_BLK - 1));
    ASSERT(get_vcount(bank, gc_vblock) == VC_MAX);
    ASSERT(!is_bad_block(bank, gc_vblock));

    // 1. load p2l list from last page offset of victim block (4B x PAGES_PER_BLK)
    // fix minor bug
    nand_page_ptread(bank, vt_vblock, PAGES_PER_BLK - 1, 0,
                     ((sizeof(UINT32) * PAGES_PER_BLK + BYTES_PER_SECTOR - 1 ) / BYTES_PER_SECTOR), FTL_BUF(bank), RETURN_WHEN_DONE);
    mem_copy(g_misc_meta[bank].lpn_list_of_cur_vblock, FTL_BUF(bank), sizeof(UINT32) * PAGES_PER_BLK);
    // 2. copy-back all valid pages to free space
    for (src_page = 0; src_page < (PAGES_PER_BLK - 1); src_page++)
    {
        // get lpn of victim block from a read lpn list
        src_lpn = get_lpn(bank, src_page);
        CHECK_VPAGE(get_vpn(src_lpn));

        // determine whether the page is valid or not
        if (get_vpn(src_lpn) !=
            ((vt_vblock * PAGES_PER_BLK) + src_page))
        {
            // invalid page
            continue;
        }
        ASSERT(get_lpn(bank, src_page) != INVALID);
        CHECK_LPAGE(src_lpn);
        // if the page is valid,
        // then do copy-back op. to free space
```

cache_hit

```
        nand_page_copyback(bank,
                           vt_vblock,
                           src_page,
                           free_vpn / PAGES_PER_BLK,
                           free_vpn % PAGES_PER_BLK);
        ASSERT((free_vpn / PAGES_PER_BLK) == gc_vblock);
        // update metadata
        set_vpn(src_lpn, free_vpn);
        set_lpn(bank, (free_vpn % PAGES_PER_BLK), src_lpn);

        free_vpn++;
    }
#if OPTION_ENABLE_ASSERT
    if (vcount == 0)
    {
        ASSERT(free_vpn == (gc_vblock * PAGES_PER_BLK));
    }
#endif
    // 3. erase victim block
    nand_block_erase(bank, vt_vblock);
    ASSERT((free_vpn % PAGES_PER_BLK) < (PAGES_PER_BLK - 2));
    ASSERT((free_vpn % PAGES_PER_BLK == vcount));

/*      uart_printf("gc page count : %d", vcount); */

    // 4. update metadata
    set_vcount(bank, vt_vblock, VC_MAX);
    set_vcount(bank, gc_vblock, vcount);
    set_new_write_vpn(bank, free_vpn); // set a free page for new write
    set_gc_vblock(bank, vt_vblock); // next free block (reserve for GC)
    dec_data_blk_cnt(bank); // decrease full block count
    /* uart_print("garbage_collection end"); */
    //uart_printf("END Data GC bank = %u", bank);
    gc[bank] += 1;
}
```

"{static void} garbage_collection(UINT32 const bank)"

= "get_vt_vblock"함수를 이용하여 GC를 해줄 Victim Block을 정합니다. 그 뒤, 해당 Block의 마지막 Page에 있는 LPN List를 읽은 뒤, DRAM에 저장되어 있는 LPN List와 비교하여 같다면 Valid Page이므로 Data를 옮겨줍니다. 만약 다르다면, Invalid Page이므로 지워줍니다. 변경된 VPN을 CMT에도 적용시켜줍니다. GC 이후에 변화된 GC Block, VCount 정보들을 갱신하여줍니다.

```c
static void garbage_collection_map(UINT32 const bank)
{
    uart_printf("Map GC bank = %u", bank);
    ASSERT(bank < NUM_BANKS);
    g_ftl_statistics[bank].gc_cnt++;

    UINT32 vt_vblock;
    UINT32 free_vpn;
    UINT32 vcount; // valid page count in victim block
    UINT32 src_page;
    UINT32 src_gtd;
    UINT32 gc_vblock;

    g_ftl_statistics[bank].gc_cnt++;

    vt_vblock = get_vt_vblock_map(bank);   // get victim block
    vcount    = get_map_vcount(bank, vt_vblock);
    gc_vblock = get_gc_map_vblock(bank);
    free_vpn  = gc_vblock * PAGES_PER_BLK;

/*     uart_printf("garbage_collection bank %d, vblock %d",bank, vt_vblock); */

    ASSERT(vt_vblock != gc_vblock);
    ASSERT(vt_vblock >= META_BLKS_PER_BANK && vt_vblock < VBLKS_PER_BANK);
    ASSERT(vcount < (PAGES_PER_BLK - 1));
    ASSERT(get_map_vcount(bank, gc_vblock) == VC_MAX);
    ASSERT(!is_bad_block(bank, gc_vblock));

    // 1. load p2l list from last page offset of victim block (4B x PAGES_PER_BLK)
    // fix minor bug
    nand_page_ptread(bank, vt_vblock, PAGES_PER_BLK - 1, 0,
                     ((sizeof(UINT32) * PAGES_PER_BLK + BYTES_PER_SECTOR - 1 ) / BYTES_PER_SECTOR), FTL_BUF(bank), RETURN_WHEN_DONE);
    mem_copy(g_misc_meta[bank].gtd_list_of_cur_vblock, FTL_BUF(bank), sizeof(UINT32) * PAGES_PER_BLK);
    // 2. copy-back all valid pages to free space
    for (src_page = 0; src_page < (PAGES_PER_BLK - 1); src_page++)
    {
        src_gtd = get_gtd(bank, src_page);
        if (src_gtd == INVALID)
            continue;
        if (read_dram_32(gtd_idx_addr(bank, src_gtd)) >= (VBLKS_PER_BANK * PAGES_PER_BLK))
        {
            uart_printf("src_gtd = %u, vpn = %u", src_gtd, read_dram_32(gtd_idx_addr(bank, src_gtd)));
            uart_printf("MAX gtd_idx = %u", (VBLKS_PER_BANK * PAGES_PER_BLK) / N_MAP_ENTRIES_PER_PAGE);
        }
        CHECK_VPAGE(read_dram_32(gtd_idx_addr(bank, src_gtd)));

        if (read_dram_32(gtd_idx_addr(bank, src_gtd)) != ((vt_vblock * PAGES_PER_BLK) + src_page))
        {
            // invalid page
            continue;
        }
        ASSERT(get_gtd(bank, src_page) != INVALID);
        ASSERT(0 < src_gtd && src_gtd < (NUM_LPAGES / (NUM_BANKS * N_MAP_ENTRIES_PER_PAGE)));
```

```
        nand_page_copyback(bank,
                           vt_vblock,
                           src_page,
                           free_vpn / PAGES_PER_BLK,
                           free_vpn % PAGES_PER_BLK);
        ASSERT((free_vpn / PAGES_PER_BLK) == gc_vblock);
        // update metadata
        set_gtd(bank, (free_vpn % PAGES_PER_BLK), src_gtd);
        CHECK_VPAGE(free_vpn);
        write_dram_32(gtd_idx_addr(bank, src_gtd), free_vpn);

        free_vpn++;
    }
#if OPTION_ENABLE_ASSERT
    if (vcount == 0)
    {
        ASSERT(free_vpn == (gc_vblock * PAGES_PER_BLK));
    }
#endif
    // 3. erase victim block
    nand_block_erase(bank, vt_vblock);
    ASSERT((free_vpn % PAGES_PER_BLK) < (PAGES_PER_BLK - 2));
    ASSERT((free_vpn % PAGES_PER_BLK == vcount));

/*      uart_printf("gc page count : %d", vcount); */

    // 4. update metadata
    set_map_vcount(bank, vt_vblock, VC_MAX);
    set_map_vcount(bank, gc_vblock, vcount);
    set_new_map_vpn(bank, free_vpn); // set a free page for new write
    set_gc_map_vblock(bank, vt_vblock); // next free block (reserve for GC)
    dec_map_blk_cnt(bank); // decrease map block count
    /* uart_print("garbage_collection end"); */
    uart_printf("END Map GC bank = %u", bank);
    map_gc += 1;
}
```

"{static void} garbage_collection_map(UINT32 const bank)"

= Data GC와 같은 형식으로 만들었습니다. LPN 대신에 GTD Index를 가지고 List를 비교하여 Valid Page와 Invalid Page를 판단합니다. 역시 Map GC 이후에 변경된 정보들을 갱신하여줍니다.

```
static UINT32 get_vcount(UINT32 const bank, UINT32 const vblock);
static UINT32 get_map_vcount(UINT32 const bank, UINT32 const vblock);
```

```
static UINT32 get_vcount(UINT32 const bank, UINT32 const vblock)
{
    UINT32 vcount;

    ASSERT(bank < NUM_BANKS);
    ASSERT((vblock >= META_BLKS_PER_BANK) && (vblock < VBLKS_PER_BANK));

    vcount = read_dram_16(VCOUNT_ADDR + (((bank * VBLKS_PER_BANK) + vblock) * sizeof(UINT16)));
    ASSERT((vcount < PAGES_PER_BLK) || (vcount == VC_MAX));

    return vcount;
}
static UINT32 get_map_vcount(UINT32 const bank, UINT32 const vblock)
{
    UINT32 vcount;

    ASSERT(bank < NUM_BANKS);
    ASSERT((vblock >= META_BLKS_PER_BANK) && (vblock < VBLKS_PER_BANK));

    vcount = read_dram_16(VCOUNT_MAP_ADDR + (((bank * VBLKS_PER_BANK) + vblock) * sizeof(UINT16)));
    ASSERT((vcount < PAGES_PER_BLK) || (vcount == VC_MAX));

    return vcount;
}
```

"{static UINT32} get_vcount(UINT32 const bank, UINT32 const bank)"

"{static UINT32} get_map_vcount(UINT32 const bank, UINT32 const vblock)"

= 기존의 "get_vcount"함수와 같은 형식으로 "get_map_vcount"함수를 만들었습니다. "get_vcount"함수는 Data Block들의 Valid Page 개수를 가져오며, "get_map_vcount"함수는 Map Block들의 Valid Page 개수를 가져옵니다.

```
static UINT32 get_vt_vblock(UINT32 const bank);
static UINT32 get_vt_vblock_map(UINT32 const bank);
```

```
static UINT32 get_vcount(UINT32 const bank, UINT32 const vblock)
{
    UINT32 vcount;

    ASSERT(bank < NUM_BANKS);
    ASSERT((vblock >= META_BLKS_PER_BANK) && (vblock < VBLKS_PER_BANK));

    vcount = read_dram_16(VCOUNT_ADDR + (((bank * VBLKS_PER_BANK) + vblock) * sizeof(UINT16)));
    ASSERT((vcount < PAGES_PER_BLK) || (vcount == VC_MAX));

    return vcount;
}
static UINT32 get_map_vcount(UINT32 const bank, UINT32 const vblock)
{
    UINT32 vcount;

    ASSERT(bank < NUM_BANKS);
    ASSERT((vblock >= META_BLKS_PER_BANK) && (vblock < VBLKS_PER_BANK));

    vcount = read_dram_16(VCOUNT_MAP_ADDR + (((bank * VBLKS_PER_BANK) + vblock) * sizeof(UINT16)));
    ASSERT((vcount < PAGES_PER_BLK) || (vcount == VC_MAX));

    return vcount;
}
```

"{static UINT32} get_vt_vblock(UINT32 const bank)"

"{static UINT32} get_vt_vblock_map(UINT32 const bank)"

= 기존의 "get_vt_vblock"함수와 같은 형식으로 "get_vt_vblock_map"함수를 만들었습니다. "get_vt_vblock"함수는 현재 사용중인 Data Block들 중에서 Valid Page가 가장 적은 Block을 선택하여 Return해줍니다. "get_vt_vblock_map" 함수 역시 같은 동작으로 Map Block들 중에서 Valid Page가 가장 적은 Block을 선택하여 Return해줍니다.

```
static UINT32 assign_new_write_vpn(UINT32 const bank);
static UINT32 assign_new_map_vpn(UINT32 const bank);
```

```
ASSERT(bank < NUM_BANKS);

UINT32 write_vpn;
UINT32 vblock;

write_vpn = get_cur_write_vpn(bank);
vblock    = write_vpn / PAGES_PER_BLK;

// NOTE: if next new write page's offset is
// the last page offset of vblock (i.e. PAGES_PER_BLK - 1),
if ((write_vpn % PAGES_PER_BLK) == (PAGES_PER_BLK - 2))
{
    // then, because of the flash controller limitation
    // (prohibit accessing a spare area (i.e. OOB)),
    // thus, we persistenly write a lpn list into last page of vblock.
    mem_copy(FTL_BUF(bank), g_misc_meta[bank].lpn_list_of_cur_vblock, sizeof(UINT32) * PAGES_PER_BLK);
    // fix minor bug
    nand_page_ptprogram(bank, vblock, PAGES_PER_BLK - 1, 0,
                        ((sizeof(UINT32) * PAGES_PER_BLK + BYTES_PER_SECTOR - 1 ) / BYTES_PER_SECTOR), FTL_BUF(bank));

    mem_set_sram(g_misc_meta[bank].lpn_list_of_cur_vblock, 0x00000000, sizeof(UINT32) * PAGES_PER_BLK);

    inc_data_blk_cnt(bank);

    // do garbage collection if necessary
    if (is_full_data_blks(bank))
    {
        if (gc[bank] == 0)
        {
            for (UINT32 blk = vblock + 1; blk < VBLKS_PER_BANK; blk++)
                set_vcount(bank, blk, VC_MAX);
        }
        garbage_collection(bank);
        return get_cur_write_vpn(bank);
    }
    do
    {
        vblock++;

        ASSERT(vblock != VBLKS_PER_BANK);
    }while (get_vcount(bank, vblock) == VC_MAX);
    write_dram_16(VCOUNT_MAP_ADDR + ((bank * VBLKS_PER_BANK) + vblock) * sizeof(UINT16), VC_MAX);
}
// write page -> next block
if (vblock != (write_vpn / PAGES_PER_BLK))
{
    write_vpn = vblock * PAGES_PER_BLK;
}
else
{
    write_vpn++;
}
set_new_write_vpn(bank, write_vpn);

return write_vpn;
```

```
ASSERT(bank < NUM_BANKS);

UINT32 write_vpn;
UINT32 vblock;

write_vpn = get_cur_map_vpn(bank);
vblock    = write_vpn / PAGES_PER_BLK;

// NOTE: if next new write page's offset is
// the last page offset of vblock (i.e. PAGES_PER_BLK - 1),
if ((write_vpn % PAGES_PER_BLK) == (PAGES_PER_BLK - 2))
{
    /*
    uart_printf("Old Map Block = %u", vblock);
    for (UINT32 page = 0; page < PAGES_PER_BLK; page++)
    {
        if (get_gtd(bank, page) != INVALID)
            uart_printf("bank = %u, blk = %u, page = %u, gtd_idx = %u", bank, vblock, page, get_gtd(bank, page));
    }*/
    mem_copy(FTL_BUF(bank), g_misc_meta[bank].gtd_list_of_cur_vblock, sizeof(UINT32) * PAGES_PER_BLK);
    // fix minor bug
    nand_page_ptprogram(bank, vblock, PAGES_PER_BLK - 1, 0,
                        ((sizeof(UINT32) * PAGES_PER_BLK + BYTES_PER_SECTOR - 1 ) / BYTES_PER_SECTOR), FTL_BUF(bank));
    mem_set_sram(g_misc_meta[bank].gtd_list_of_cur_vblock, 0x00000000, sizeof(UINT32) * PAGES_PER_BLK);

    inc_map_blk_cnt(bank);

    // do garbage collection if necessary
    if (is_full_map_blks(bank))
    {
        garbage_collection_map(bank);
        return get_cur_map_vpn(bank);
    }
    do
    {
        vblock++;

        ASSERT(vblock != VBLKS_PER_BANK);
    }while (get_map_vcount(bank, vblock) == VC_MAX);
    //uart_printf("Assign New Map Block = %u", vblock);
    write_dram_16(VCOUNT_ADDR + ((bank * VBLKS_PER_BANK) + vblock) * sizeof(UINT16), VC_MAX);
}
// write page -> next block
if (vblock != (write_vpn / PAGES_PER_BLK))
{
    write_vpn = vblock * PAGES_PER_BLK;
}
else
{
    write_vpn++;
}
CHECK_VPAGE(write_vpn);
set_new_map_vpn(bank, write_vpn);

return write_vpn;
```

"{static UINT32} assign_new_write_vpn(UINT32 const bank)"

"{static UINT32} assign_new_map_vpn(UINT32 const bank)"

= 위에 있는 함수는 "assign_new_write_vpn"이며, 아래에 있는 함수는 "assign_new_map_vpn"입니다. 기존의 "assign_new_write_vpn"함수와 같은 형식으로 "assign_new_map_vpn"함수를 만들었습니다. "assign_new_map_vpn" 함수는 "cur_write_vpn"을 증가시켜주며 다음 Page가 Block의 마지막 Page라면 LPN List를 적어준 뒤, 빈 Block을 검색하여 새롭게 할당받습니다. 만약 Data Block을 최대로 사용중이라면, Data GC를 해줍니다.

"assign_new_map_vpn"도 역시 같은 동작을 하며, "cur_map_vpn"을 증가시켜준다는 점과, GTD List, Map Block을 사용한다는 것만 다릅니다.

```
static void    set_vpn(UINT32 const lpn, UINT32 const vpn);
```

```
static void set_vpn(UINT32 const lpn, UINT32 const vpn)
{
    UINT32 bank, cache_slot, entry_num;
    UINT32 blk, page;
    if (debug)
        uart_printf("set_vpn");
    CHECK_LPAGE(lpn);
    CHECK_VPAGE(vpn);

    bank        = get_num_bank(lpn);
    cache_slot  = check_cmt(bank, lpn);
    entry_num   = get_entry_num(lpn);
    ASSERT(vpn >= (META_BLKS_PER_BANK * PAGES_PER_BLK) && vpn < (VBLKS_PER_BANK * PAGES_PER_BLK));

    if (debug)
        uart_printf("set_vpn = %u, cache_slot = %u, entry_num = %u, lpn = %u", vpn, cache_slot, entry_num, lpn);
    if (read_dram_32(cmt_entry_addr(bank, cache_slot, entry_num)) != NULL)
    {
        extract_vpn(read_dram_32(cmt_entry_addr(bank, cache_slot, entry_num)), &blk, &page);
    }
    write_dram_32(cmt_entry_addr(bank, cache_slot, entry_num), vpn);
    write_dram_32(cmt_lpn_addr(bank, cache_slot), (lpn / (NUM_BANKS * N_MAP_ENTRIES_PER_PAGE)) * (NUM_BANKS * N_MAP_ENTRIES_PER_PAGE));
    write_dram_32(cmt_dirty_addr(bank, cache_slot), 1);
    write_dram_32(cmt_time_addr(bank, cache_slot), bank_timer[bank]);
    bank_timer[bank] += 1;
    write_dram_32(PAGE_MAP_ADDR + lpn * sizeof(UINT32), vpn);
}
```

"{static void} set_vpn(UINT32 const lpn, UINT32 const vpn)"

= "check_cmt"함수를 이용하여, CMT에서 LPN이 들어갈 Cache를 검색한 뒤, Entry Number를 계산하여 입력하여줍니다. Dirty Bit를 바꿔주며, Time(or Age)를 갱신하여줍니다.

```
static UINT32 get_vpn(UINT32 const lpn);
```

```
static UINT32 get_vpn(UINT32 const lpn)
{
    UINT32 bank, cache_slot, entry_num;
    CHECK_LPAGE(lpn);

    bank        = get_num_bank(lpn);
    cache_slot  = check_cmt(bank, lpn);
    entry_num   = get_entry_num(lpn);

    return read_dram_32(cmt_entry_addr(bank, cache_slot, entry_num));
}
```

"{static UINT32} get_vpn(UINT32 const lpn)"

= "check_cmt"함수를 이용하여, CMT에서 LPN이 있는 Cache를 검색한 뒤, Entry Number를 계산하여 해당 LPN이 저장되어 있는 VPN값을 Return하여줍니다.

```
static void extract_vpn(UINT32 const gtd_vpn, UINT32* gtd_blk, UINT32* gtd_page);
static UINT32 victim_cache(UINT32 const bank);
static UINT32 LRU(UINT32 const bank, UINT32 const lpn);
static UINT32 check_cmt(UINT32 const bank, UINT32 const lpn);
static void map_read(UINT32 const bank, UINT32 const gtd_idx, UINT32 const cache_slot);
static void map_write(UINT32 const bank, UINT32 const gtd_idx, UINT32 const cache_slot);
```

```
static void extract_vpn(UINT32 const gtd_vpn, UINT32* gtd_blk, UINT32* gtd_page)
{
    *gtd_blk    = (gtd_vpn % PAGES_PER_BANK) / PAGES_PER_BLK;
    *gtd_page   = gtd_vpn % PAGES_PER_BLK;
}
```

"{static void} extract_vpn(UINT32 const gtd_vpn, UINT32* gtd_blk, UINT32* gtd_page)"

= 입력 받은 "gtd_vpn"으로부터 해당 VPN의 Block Number와 Page Number를 계산하여 값을 저장해줍니다.

```
static UINT32 victim_cache(UINT32 const bank)
{
    UINT32 cache_slot;
    UINT32 cmt_time;
    UINT32 idx = 0, min = MAX_VALUE;

    for (cache_slot = 0; cache_slot < N_CACHED_MAP_PAGE_PB; cache_slot++)
    {
        cmt_time = read_dram_32(cmt_time_addr(bank, cache_slot));
        if (cmt_time < min)
        {
            idx = cache_slot;
            min = cmt_time;
        }
    }
    return idx;
}
```

"{static UINT32} victim_cache(UINT32 const bank)"

= 모든 Cache를 확인하여 Time(or Age)값이 가장 적은 Cache를 선정하여 해당 Cache를 Victim Cache로 정하여 Return해줍니다.

```
static UINT32 LRU(UINT32 const bank, UINT32 const lpn)
{
    UINT32 cache_slot;
    UINT32 cmt_lpn, gtd_idx;

    cache_slot  = victim_cache(bank);
    cmt_lpn     = read_dram_32(cmt_lpn_addr(bank, cache_slot));
    if (read_dram_32(cmt_dirty_addr(bank, cache_slot)) == 1)
    {
        gtd_idx = get_gtd_idx(read_dram_32(cmt_lpn_addr(bank, cache_slot)));
        map_write(bank, gtd_idx, cache_slot);
    }
    gtd_idx = get_gtd_idx(lpn);
    map_read(bank, gtd_idx, cache_slot);
    return cache_slot;
}
```

"{static UINT32} LRU(UINT32 const bank, UINT32 const lpn)"

= Victim Cache를 "victim_cache"함수를 통해 받은 뒤, 해당 Cache가 Dirty 상태라면 Map Write를 해주고, 입력 받은 LPN이 있는 Map Page를 Map Read하여줍니다.

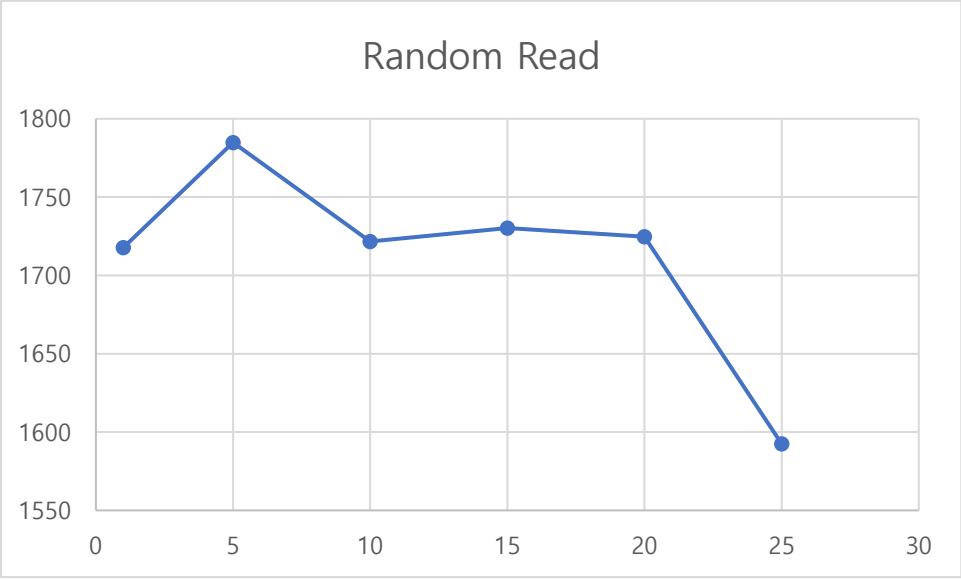
```
static UINT32 check_cmt(UINT32 const bank, UINT32 const lpn)
{
    UINT32 cache_slot, entry_num;
    UINT32 cmt_lpn, cmt_vpn;

    for (cache_slot = 0; cache_slot < N_CACHED_MAP_PAGE_PB; cache_slot++)
    {
        cmt_lpn = read_dram_32(cmt_lpn_addr(bank, cache_slot));
        if (cmt_lpn == (lpn / (NUM_BANKS * N_MAP_ENTRIES_PER_PAGE)) * (NUM_BANKS * N_MAP_ENTRIES_PER_PAGE))
        {
            entry_num = get_entry_num(lpn);
            cmt_vpn = read_dram_32(cmt_entry_addr(bank, cache_slot, entry_num));
            if (cmt_vpn == NULL)
                cache_miss++;
            else
                cache_hit++;
            return cache_slot;
        }
    }
    cache_miss++;
    return LRU(bank, lpn);
}
```

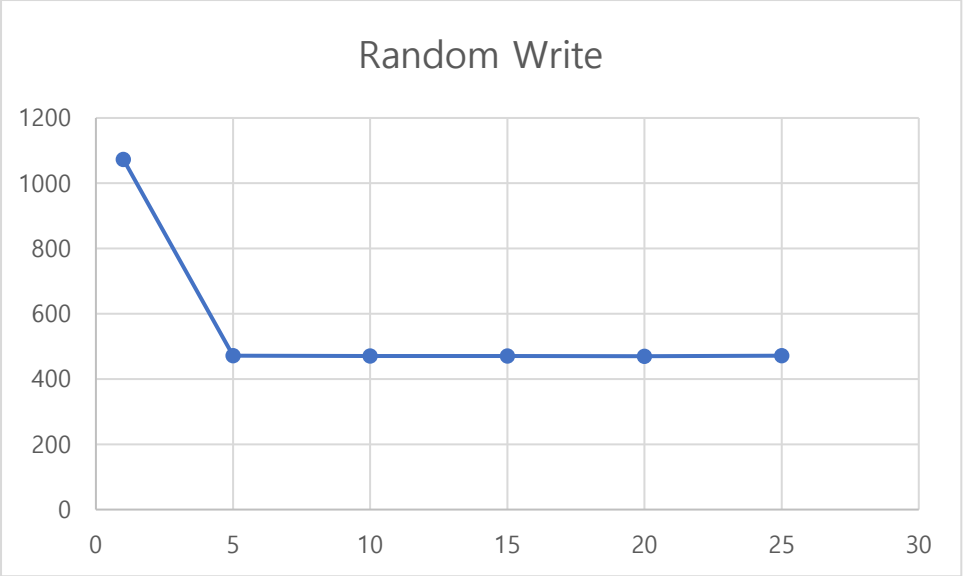
"{static UINT32} check_cmt(UINT32 const bank, UINT32 const lpn)"

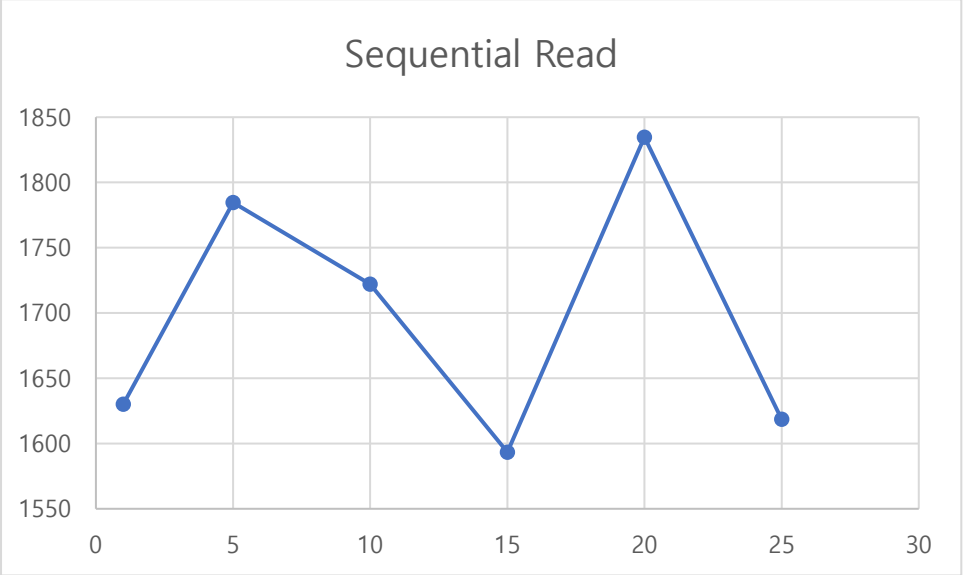
= 모든 Cache를 확인하며, 입력 받은 LPN이 포함되는 Cache가 있는지 확인하여 주며, 없다면 "LRU"함수를 이용하여 LPN이 포함되는 Cache를 읽거나 만들어줍니다.

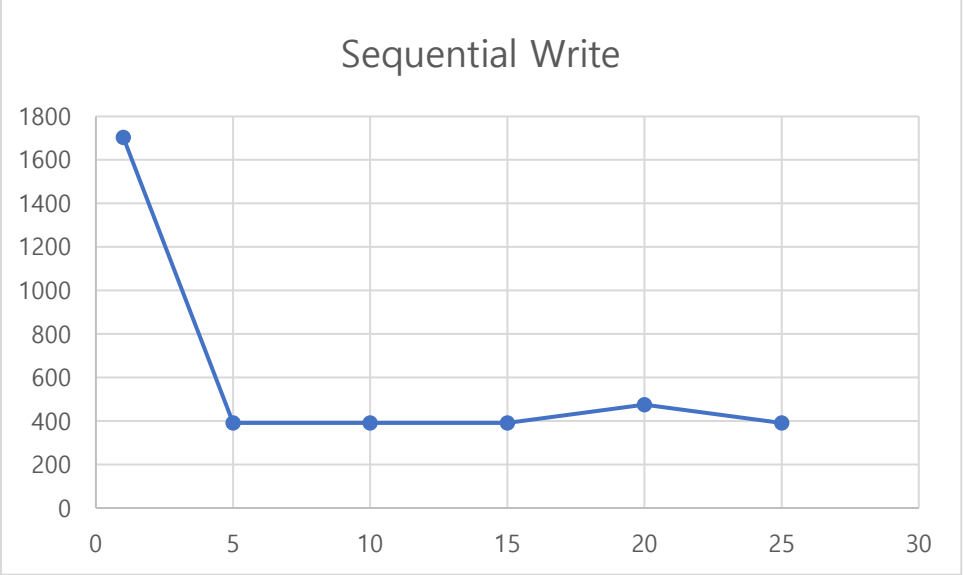
2.Analyze the Performance vs. CMT Ratio

IOmeter를 이용하여, 5분동안 진행한 결과입니다.









현재 결과 Graph를 분석하도록 하겠습니다.

우선, Random vs. Sequential을 비교하도록 하겠습니다.

두 그래프를 비교하였을 때에, Cache Ratio = 1 일 때에 이상적인 Graph가 나왔지만,

Cache Ratio가 증가함에 따라 오히려 Sequential Write가 Random Write보다 느려지는 것을 볼 수 있습니다.

또한, Random Read가 Sequential Read보다 더 빠른 곳이 존재하는 곳도 보입니다.

이상적인 결과를 나타내기에 테스트 시간이 너무 짧았다는 결론을 내렸습니다.

Read vs. Write를 비교하도록 하겠습니다.

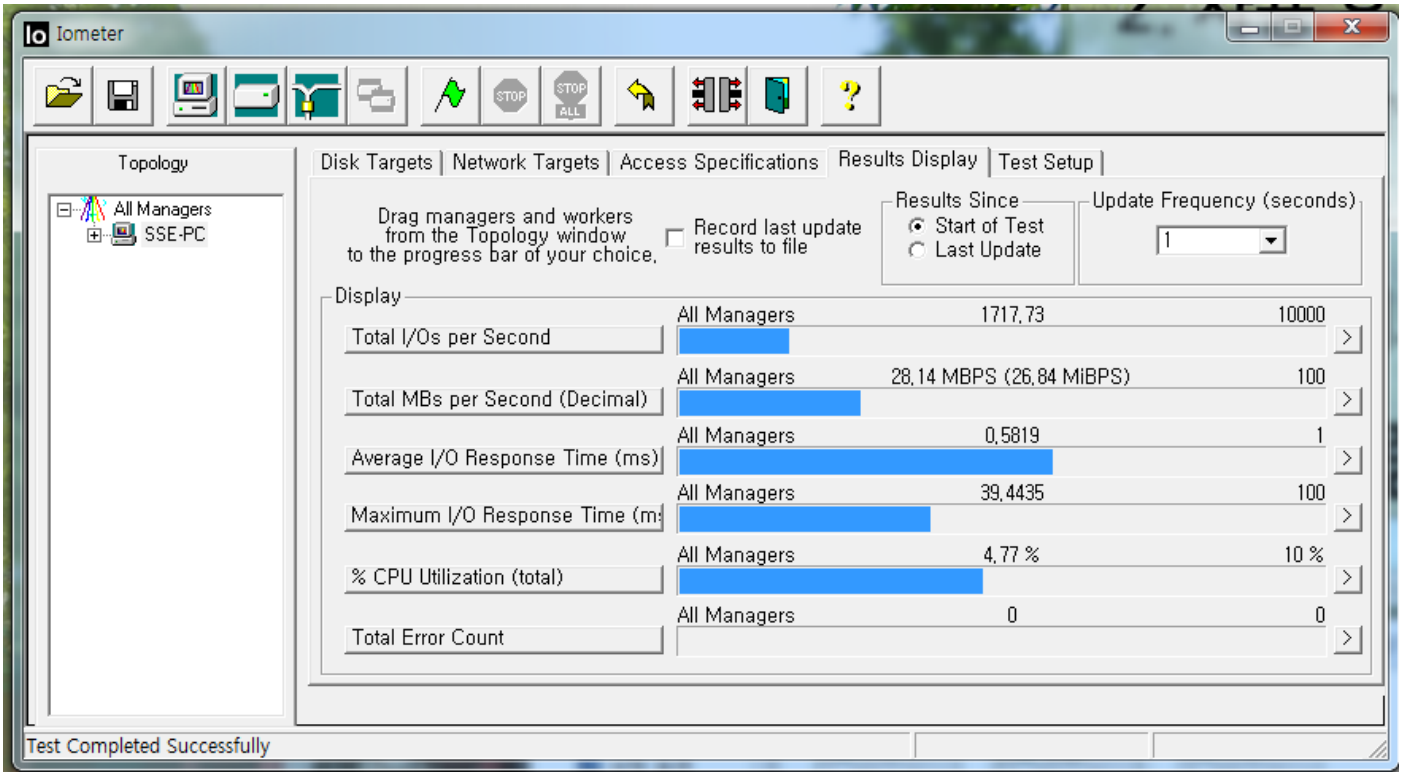
Read의 경우, CMT Ratio가 변화함에 따라 감소하는 추세를 보이지만, 중간 중간 변동이 많이 나타났습니다.

Write의 경우, CMT Ratio = 1 일 때에 가장 높은 값을 보인 뒤, 급감한 뒤 변동이 별로 없었습니다.

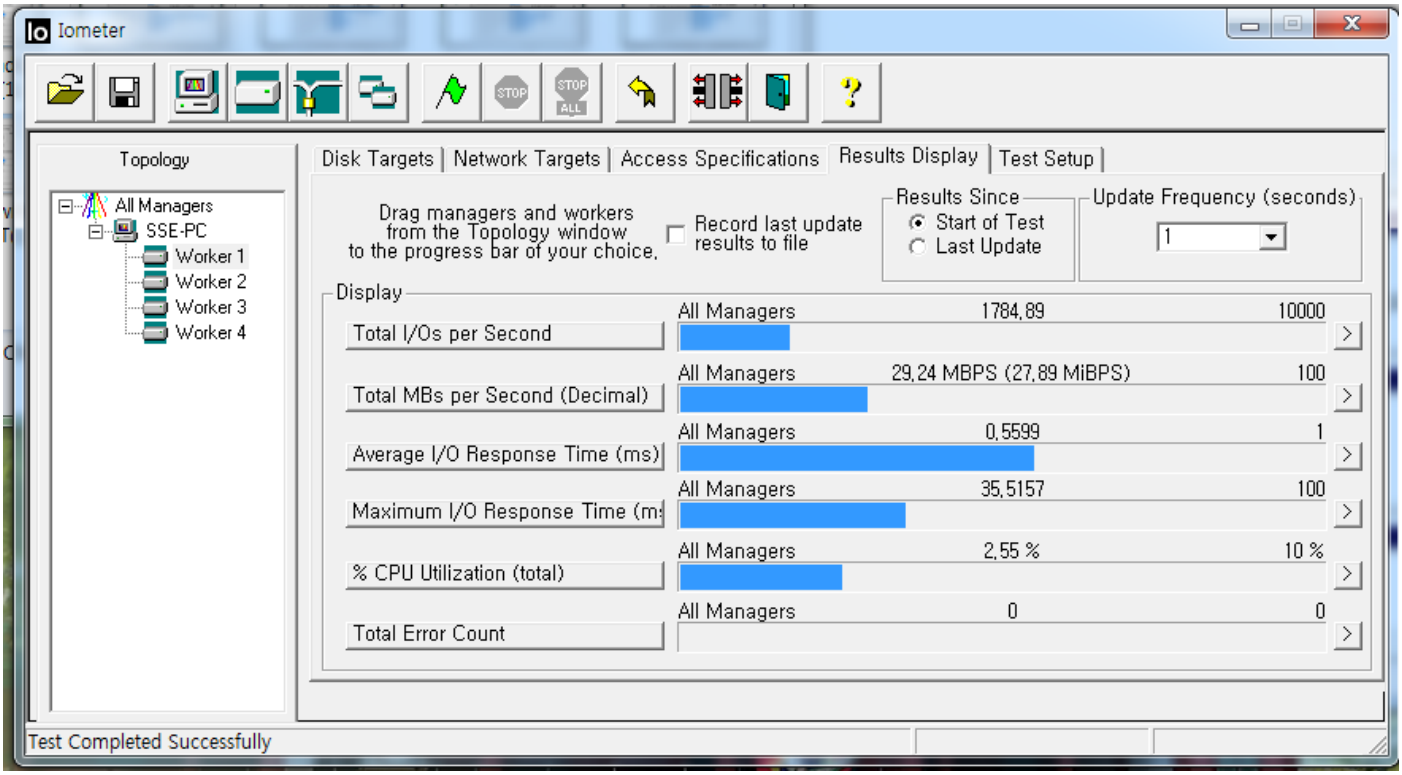
Read는 Cache의 Hit와 Miss에 따라서 속도가 많이 변화하는 것으로 생각되며,

Write는 Nand Write 뿐 아니라 Data GC와 Map GC를 동반하기 때문에 CMT가 커지더라도 속도가 수렴한다는 결론을 내렸습니다.

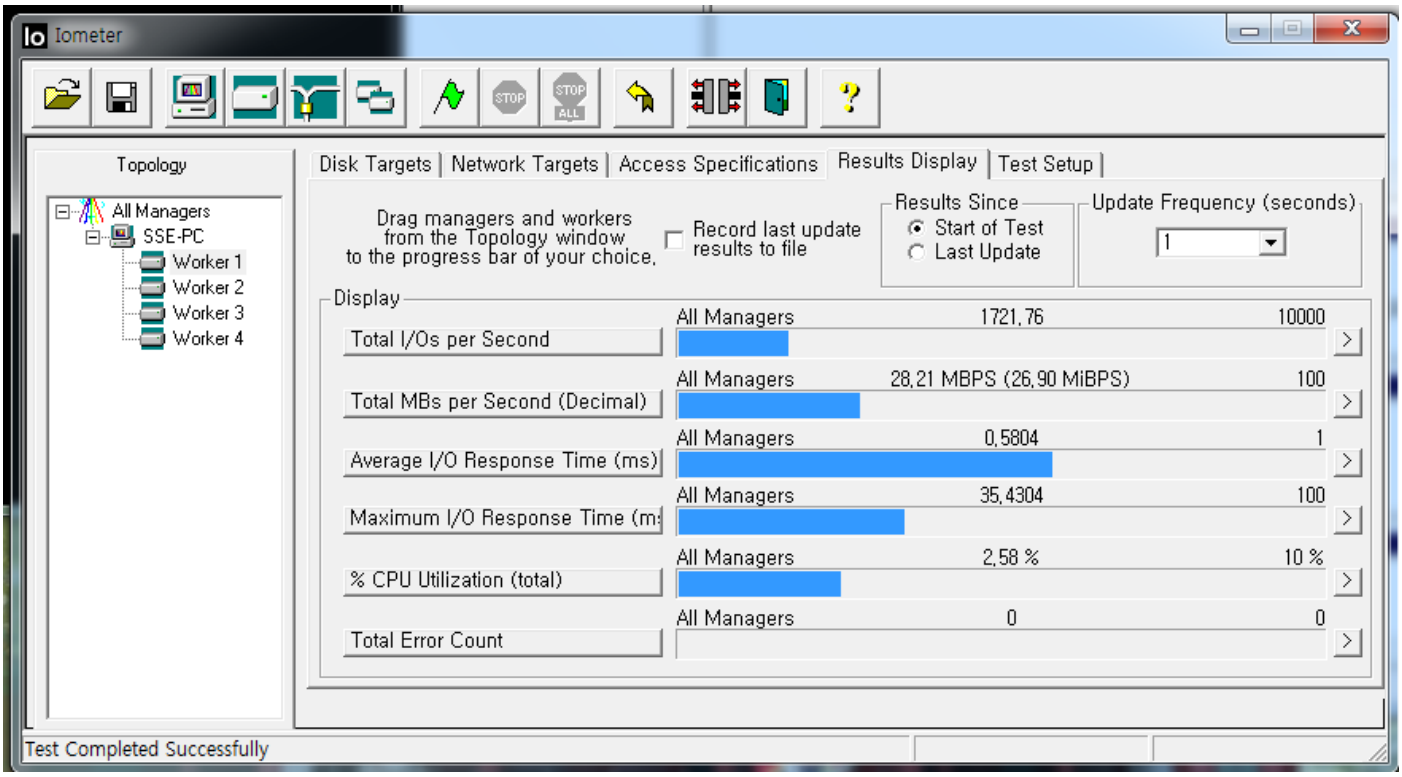
3.Measure the Performance with an IOmeter



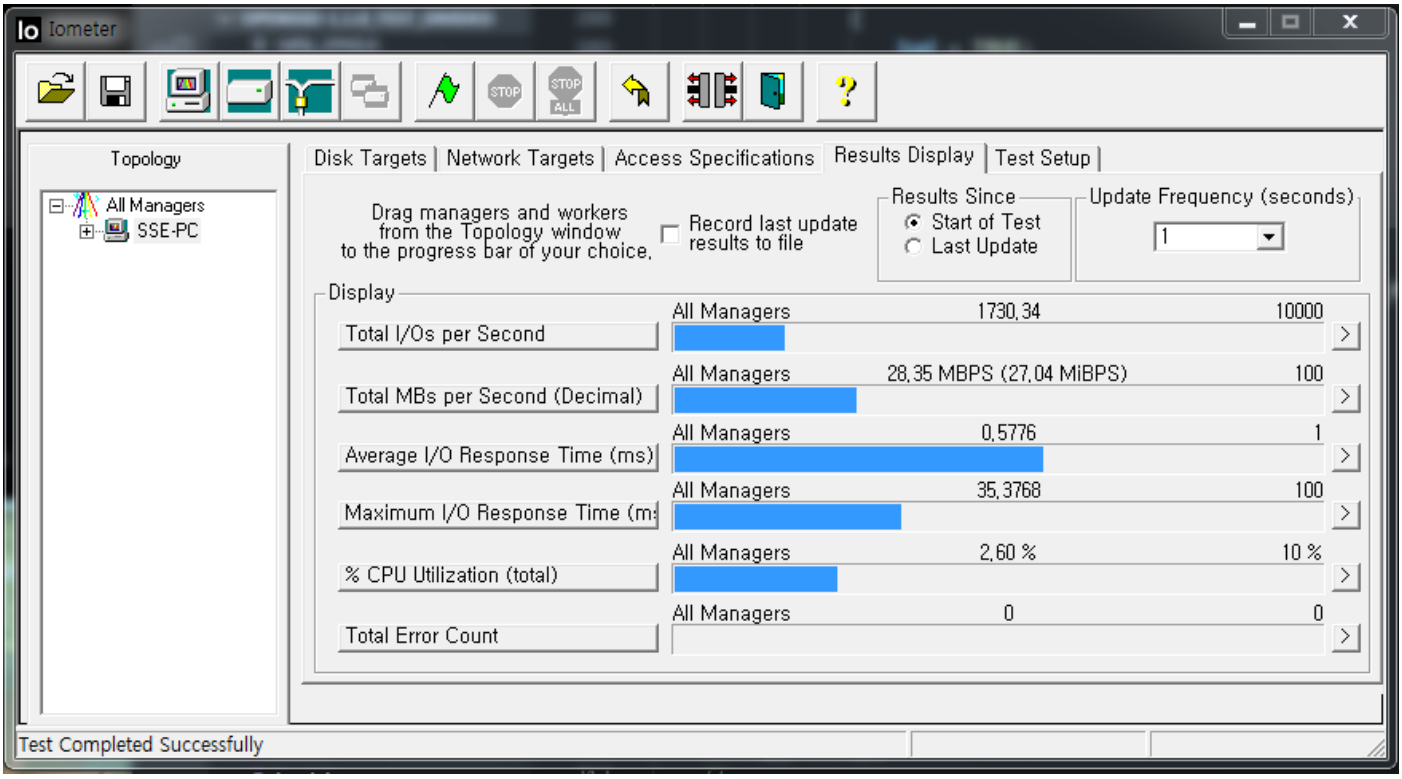
Random_read(16KB) with CMT_Ratio = 1



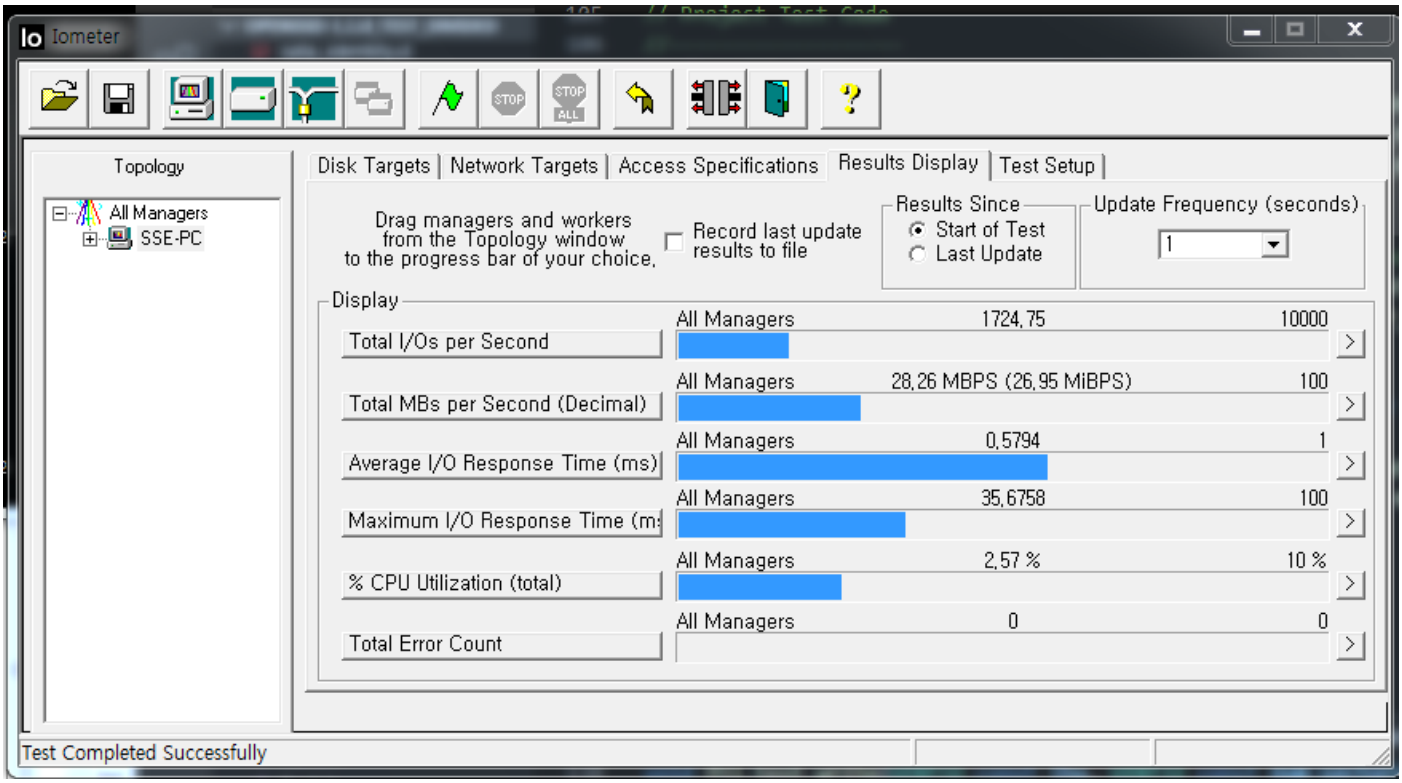
Random_read(16KB) with CMT_Ratio = 5



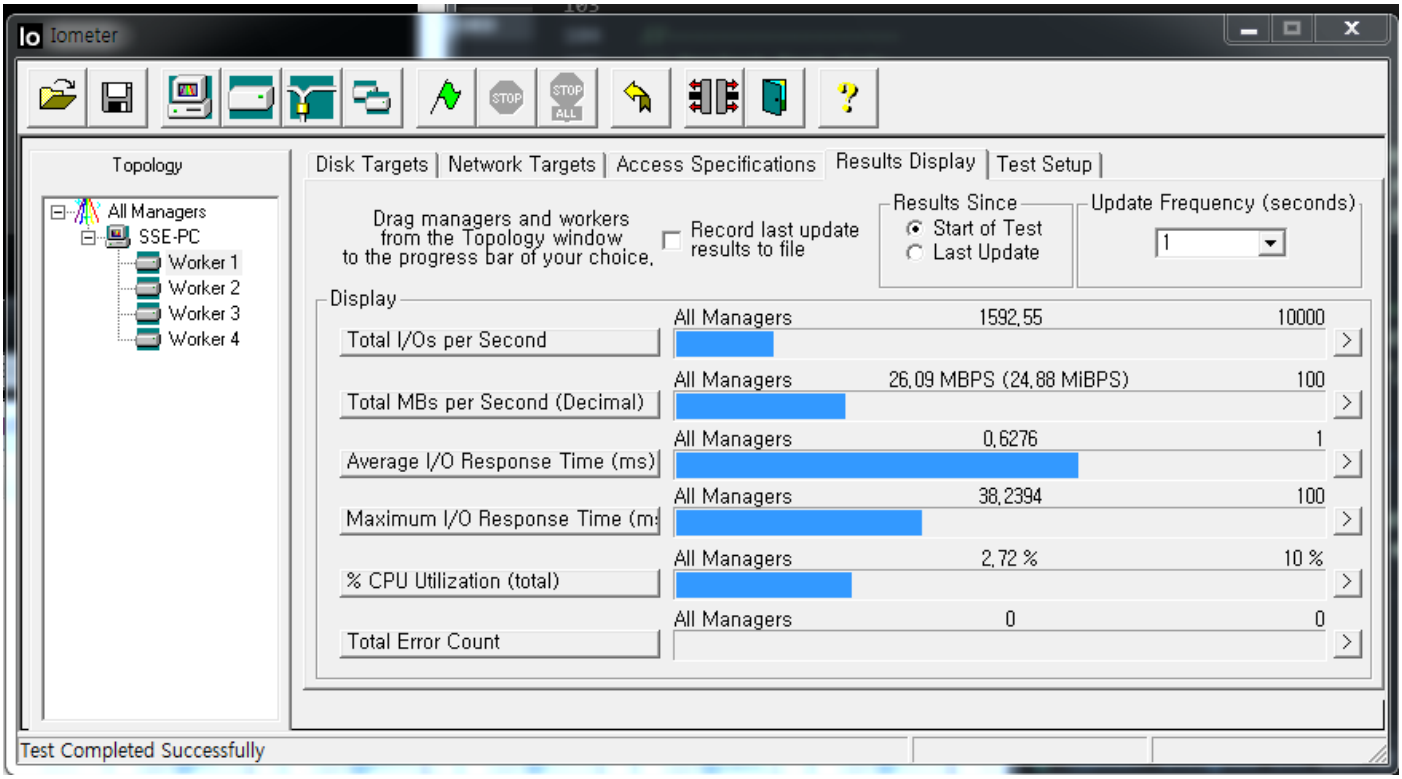
Random_read(16KB) with CMT_Ratio = 10



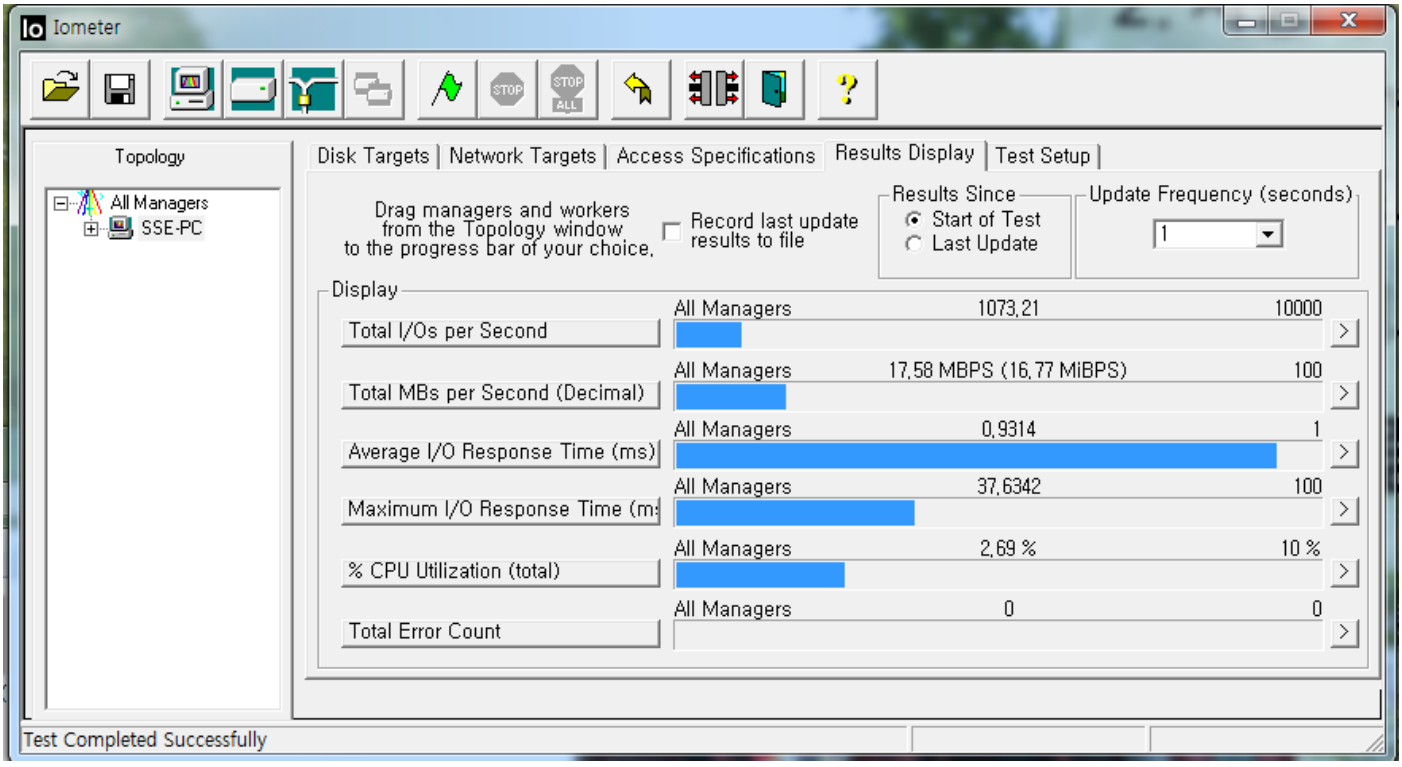
Random_read(16KB) with CMT_Ratio = 15



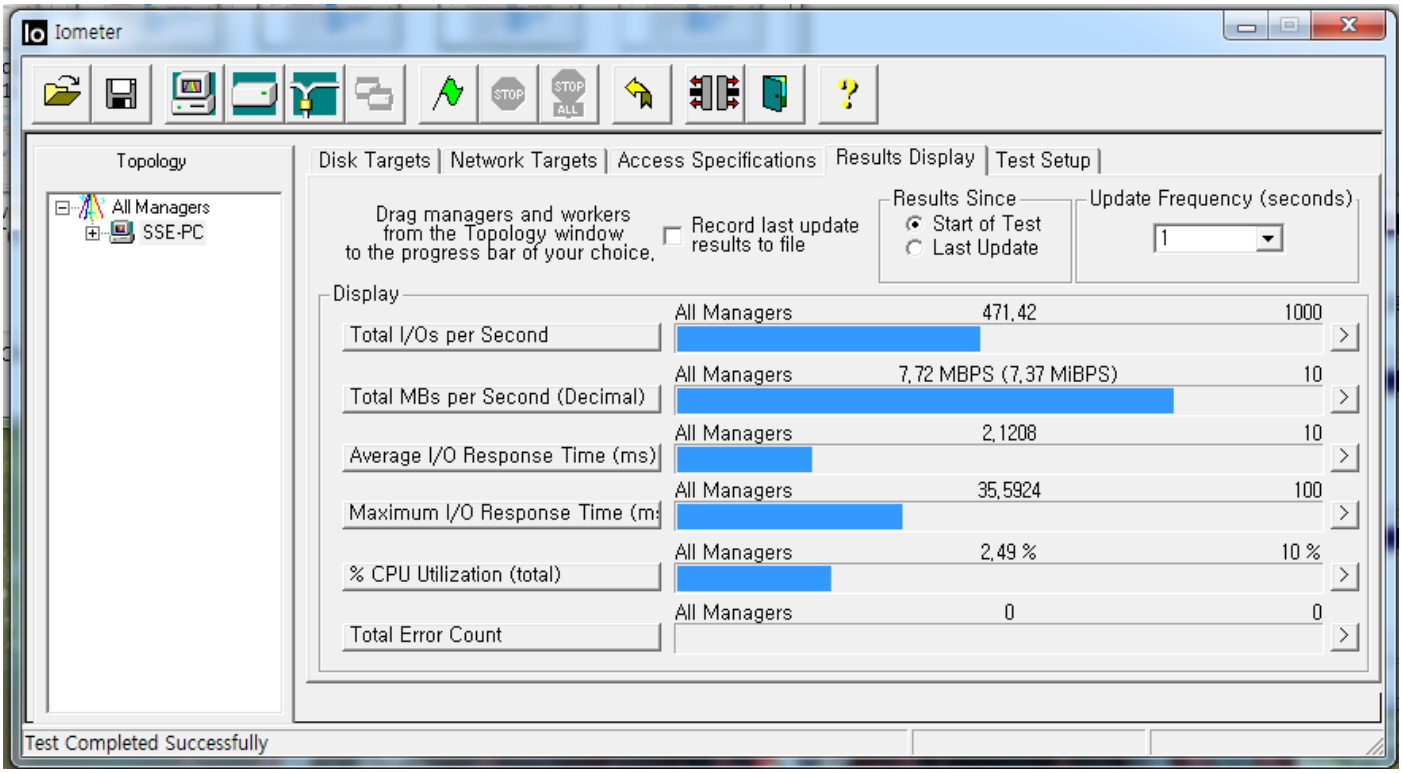
Random_read(16KB) with CMT_Ratio = 20



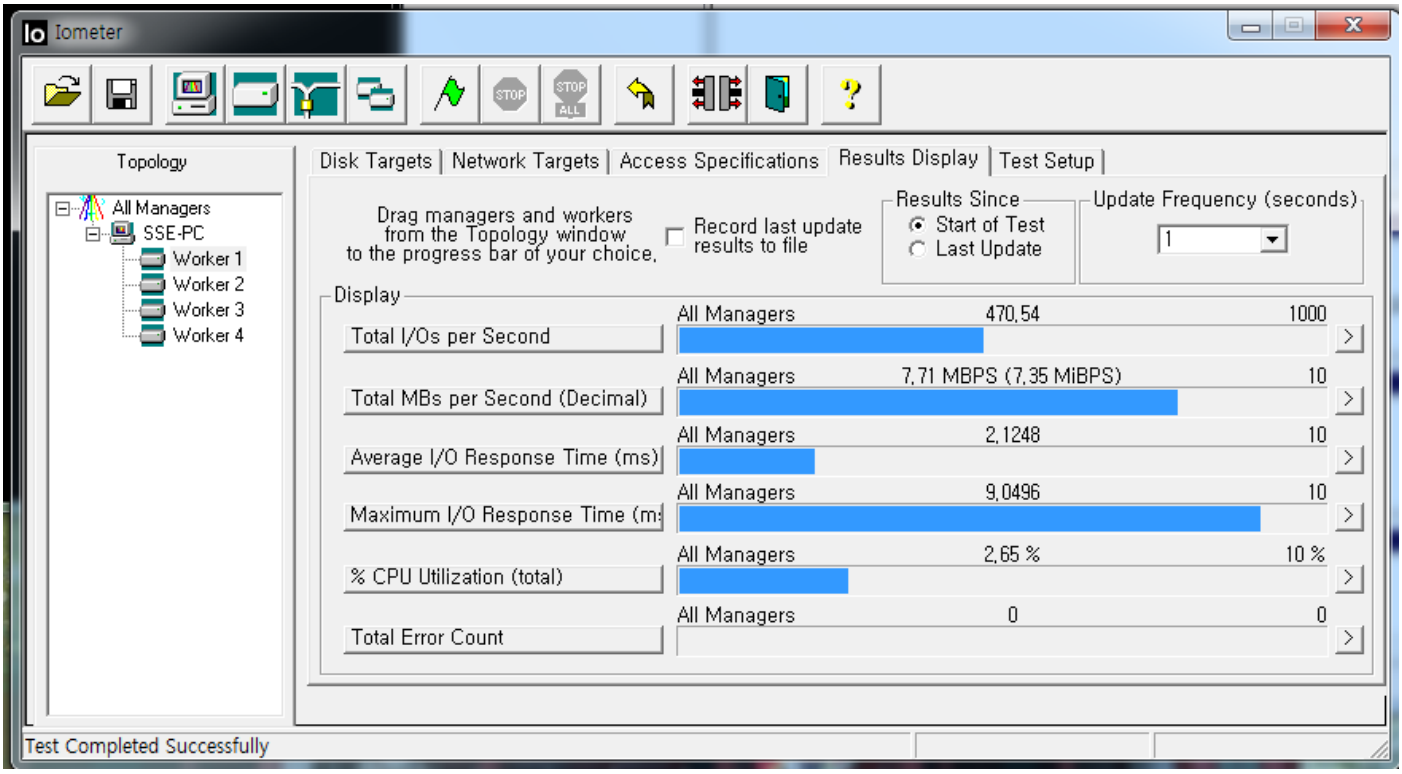
Random_read(16KB) with CMT_Ratio = 25



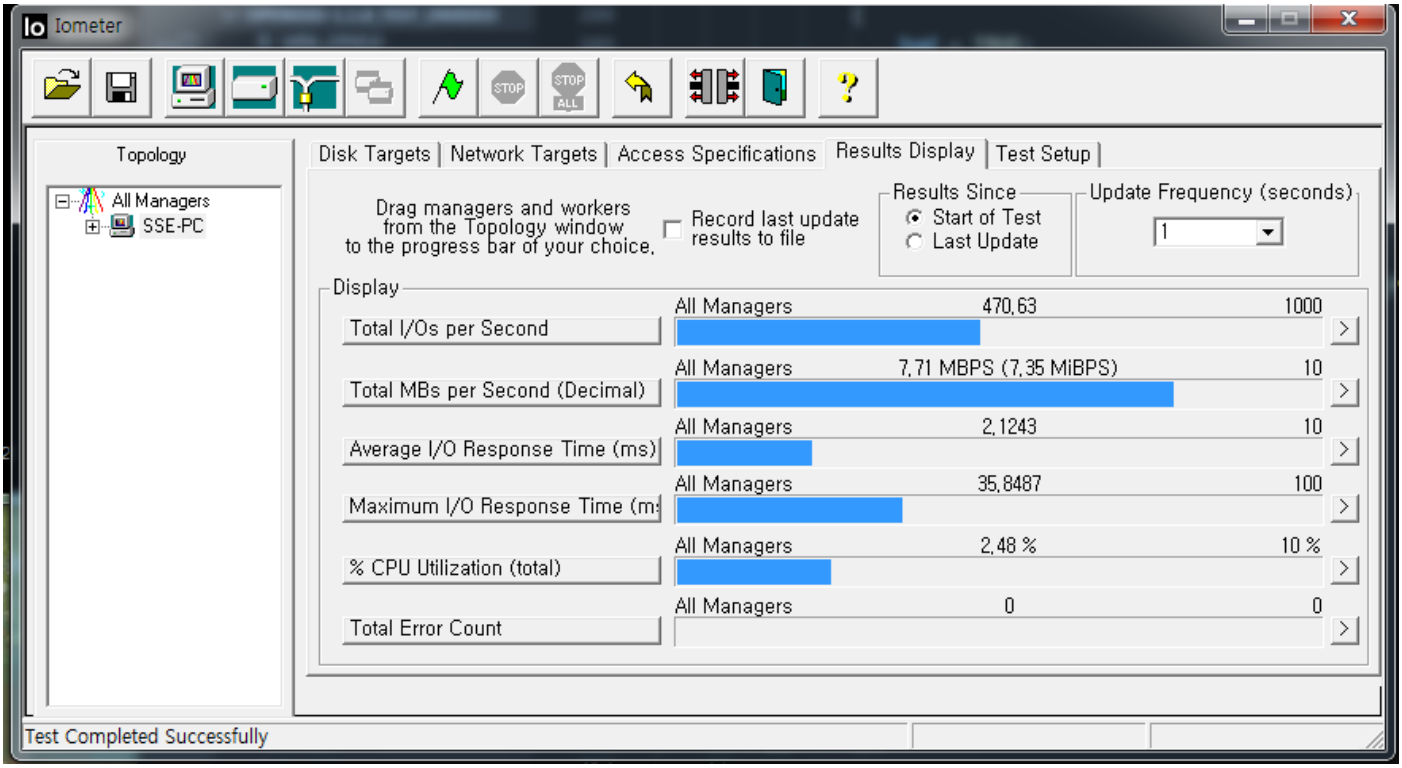
Random_write(16KB) with CMT_Ratio = 1



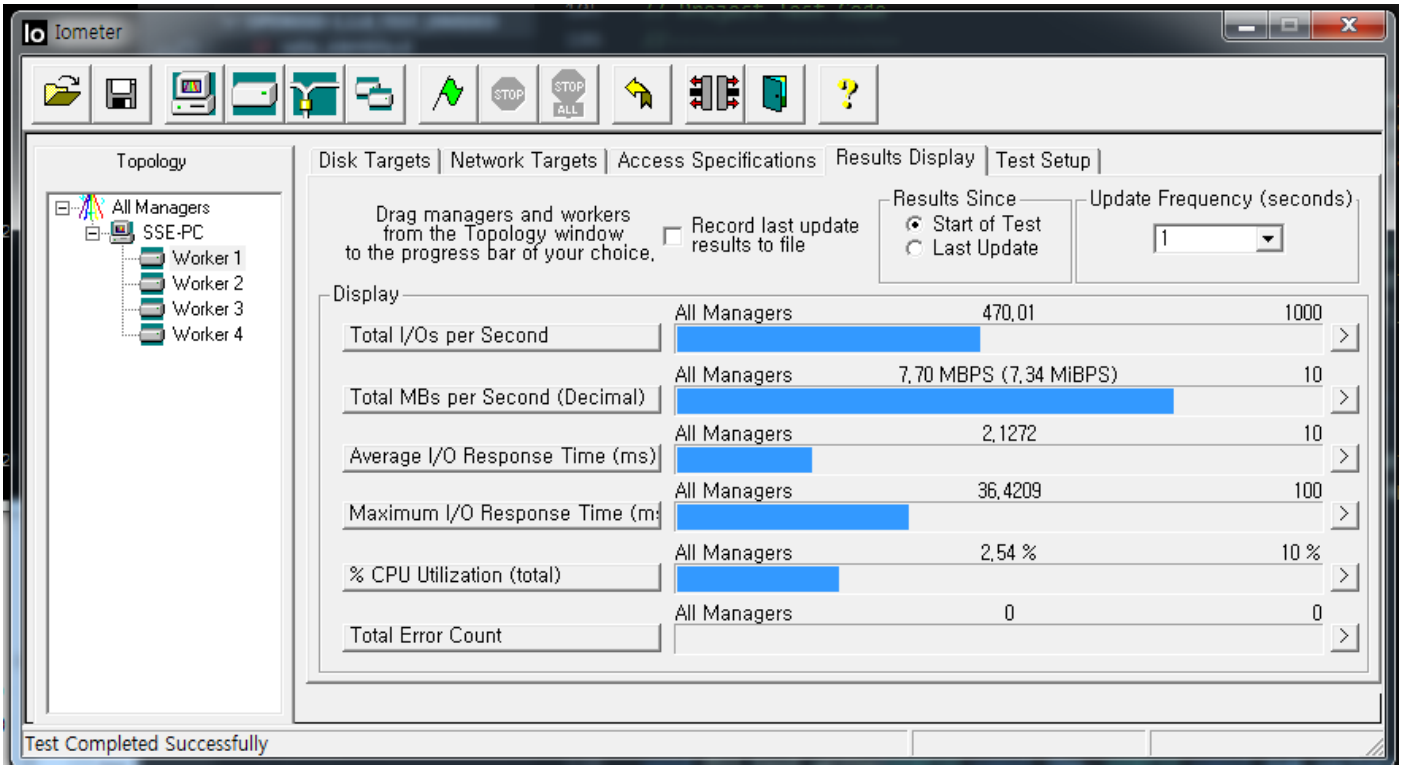
Random_write(16KB) with CMT_Ratio = 5



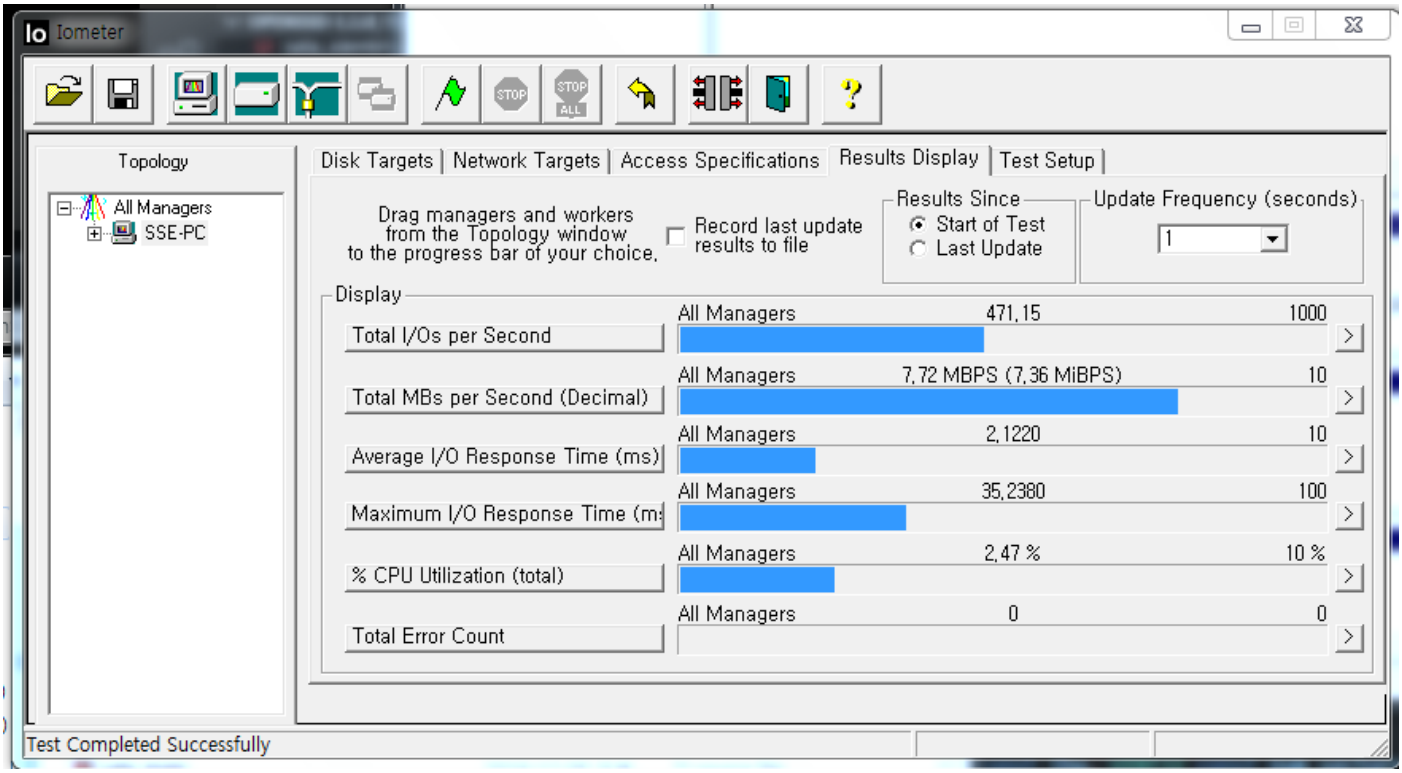
Random_write(16KB) with CMT_Ratio = 10



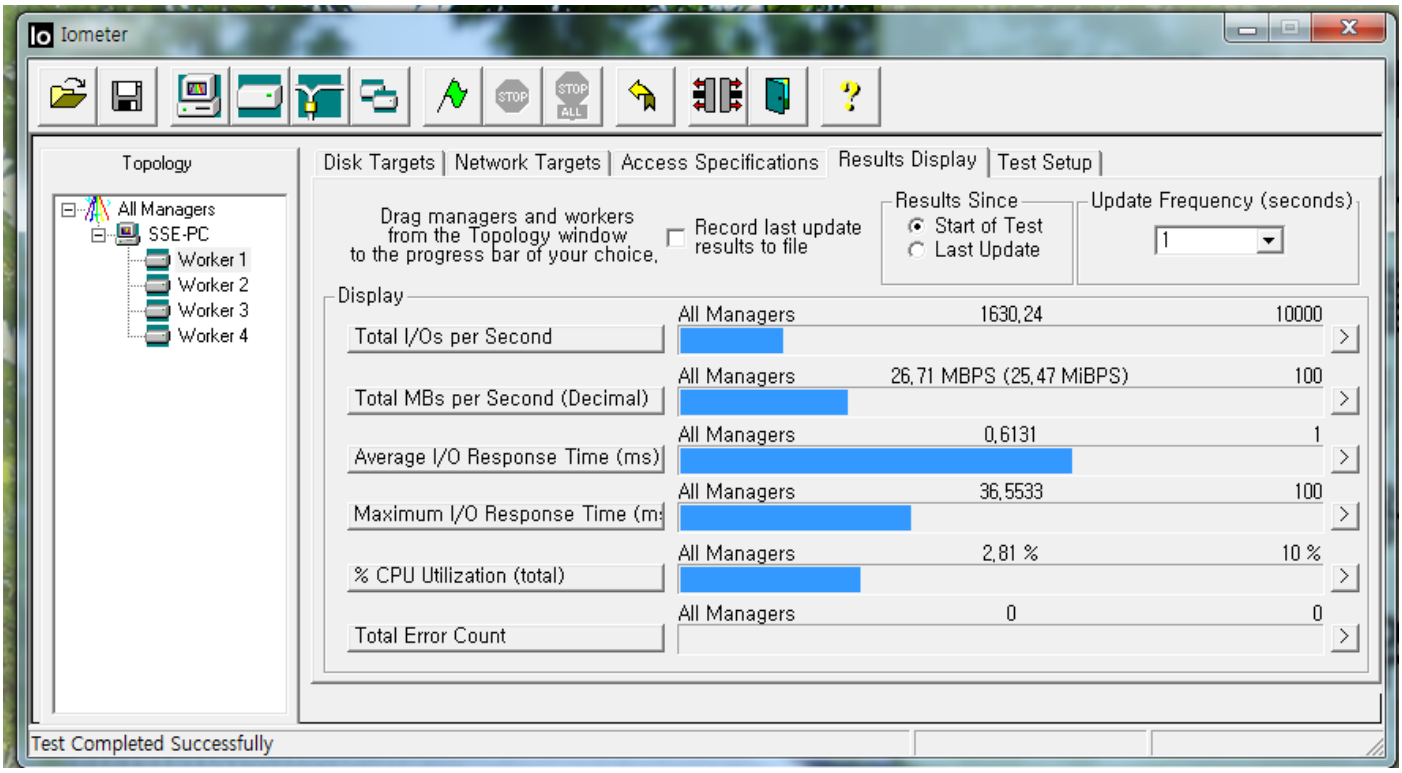
Random_write(16KB) with CMT_Ratio = 15



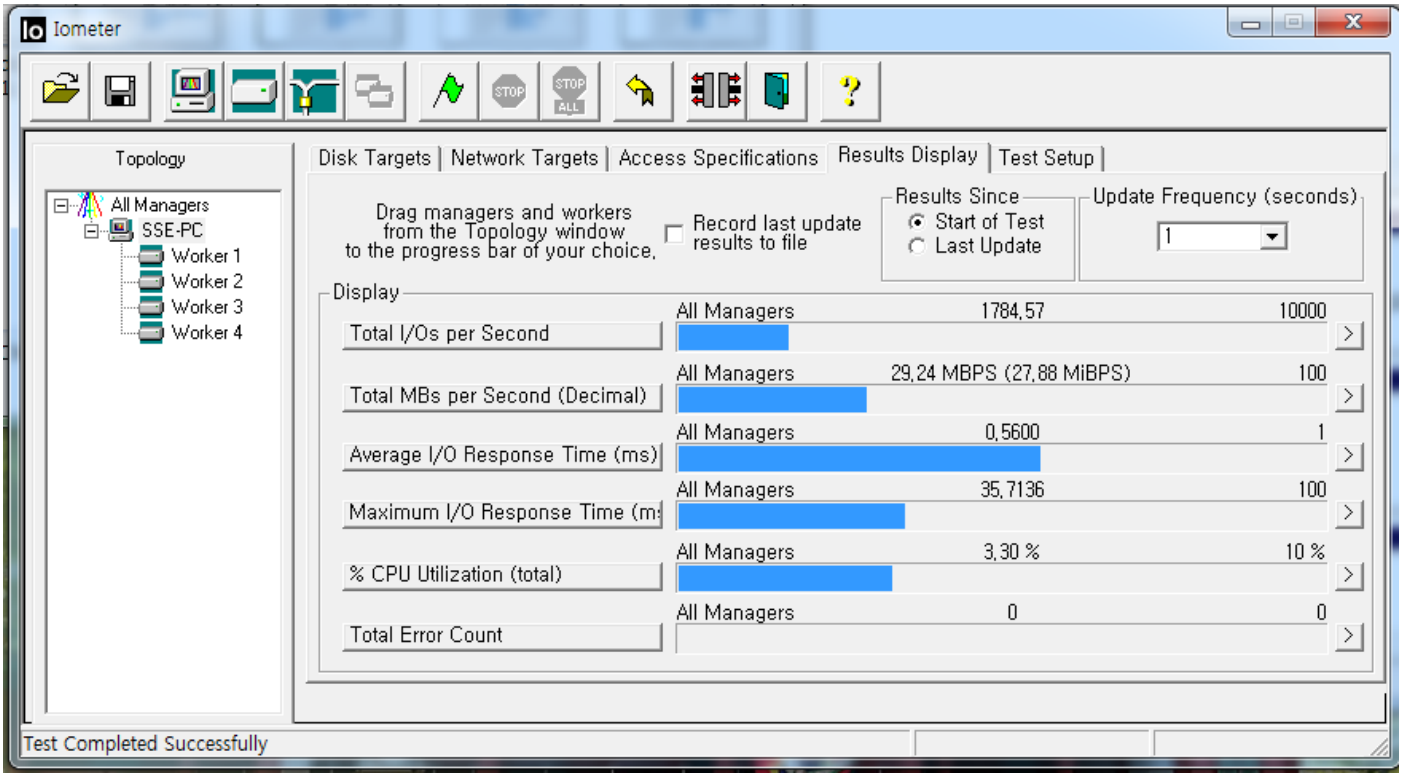
Random_write(16KB) with CMT_Ratio = 20



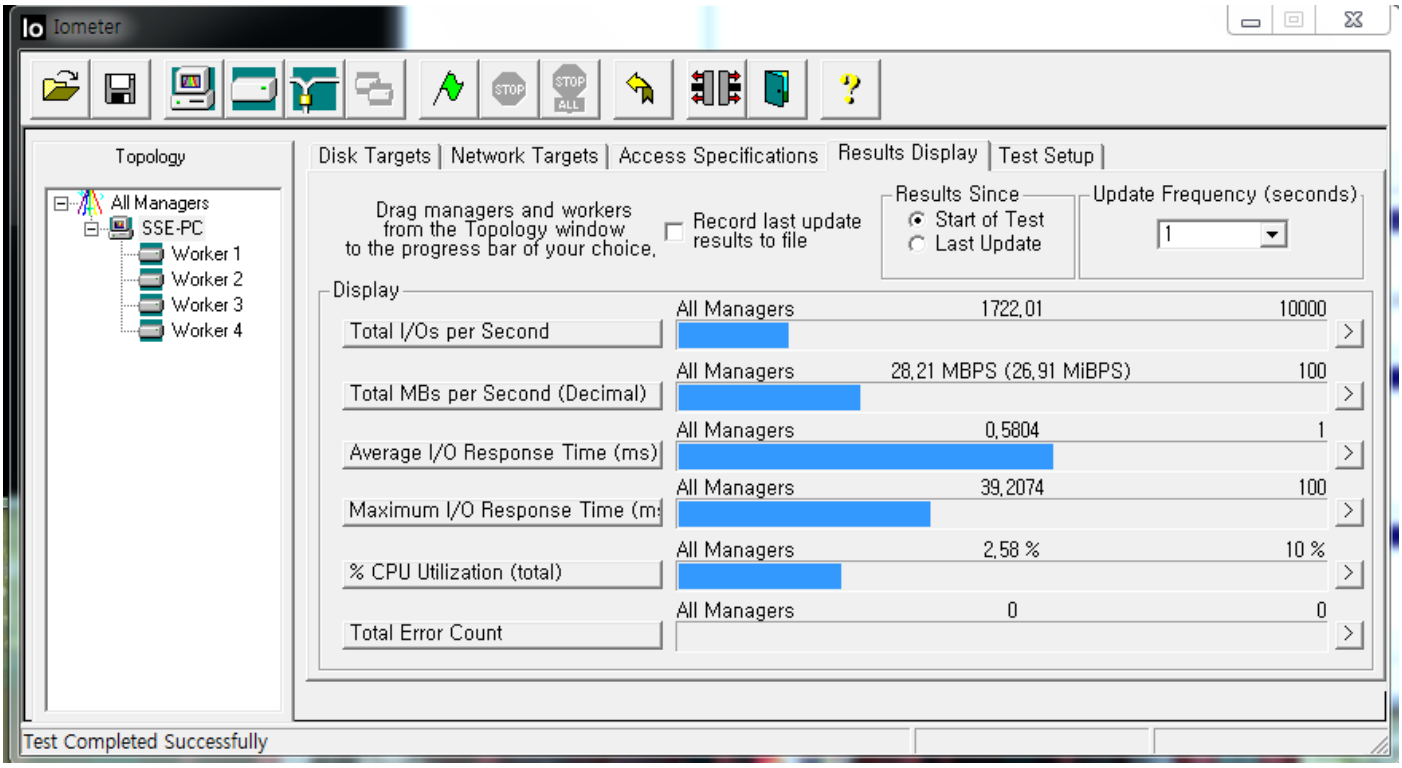
Random_write(16KB) with CMT_Ratio = 25



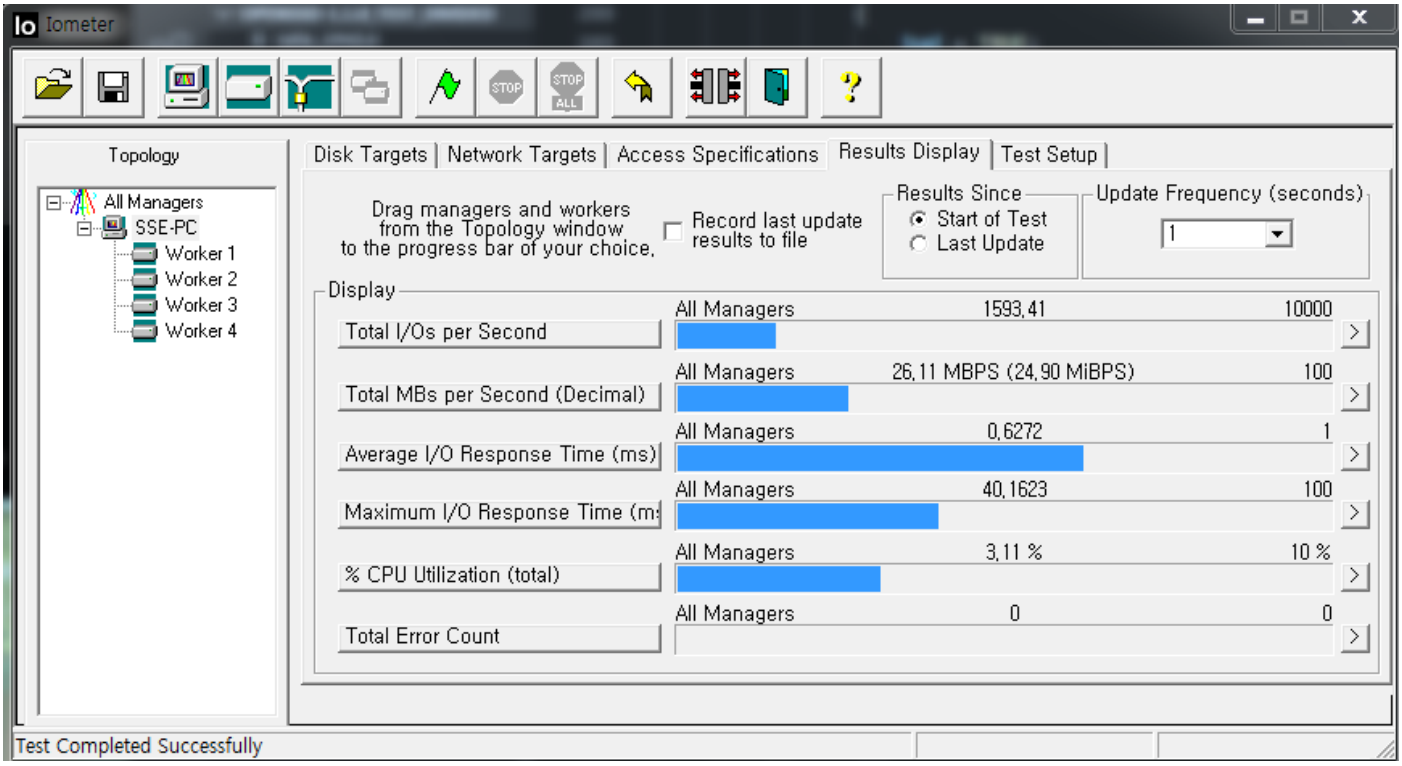
Sequential_read(16KB) with CMT_Ratio = 1



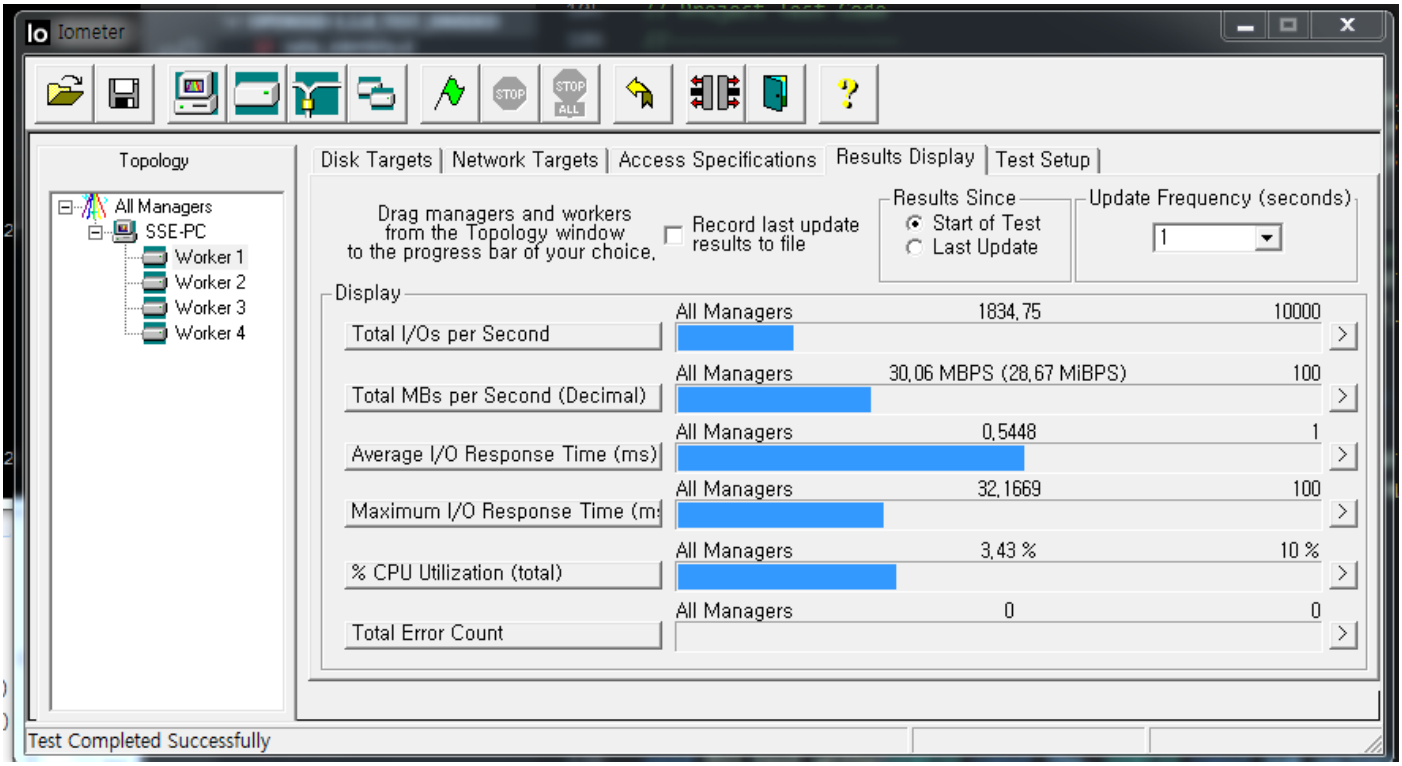
Sequential_read(16KB) with CMT_Ratio = 5



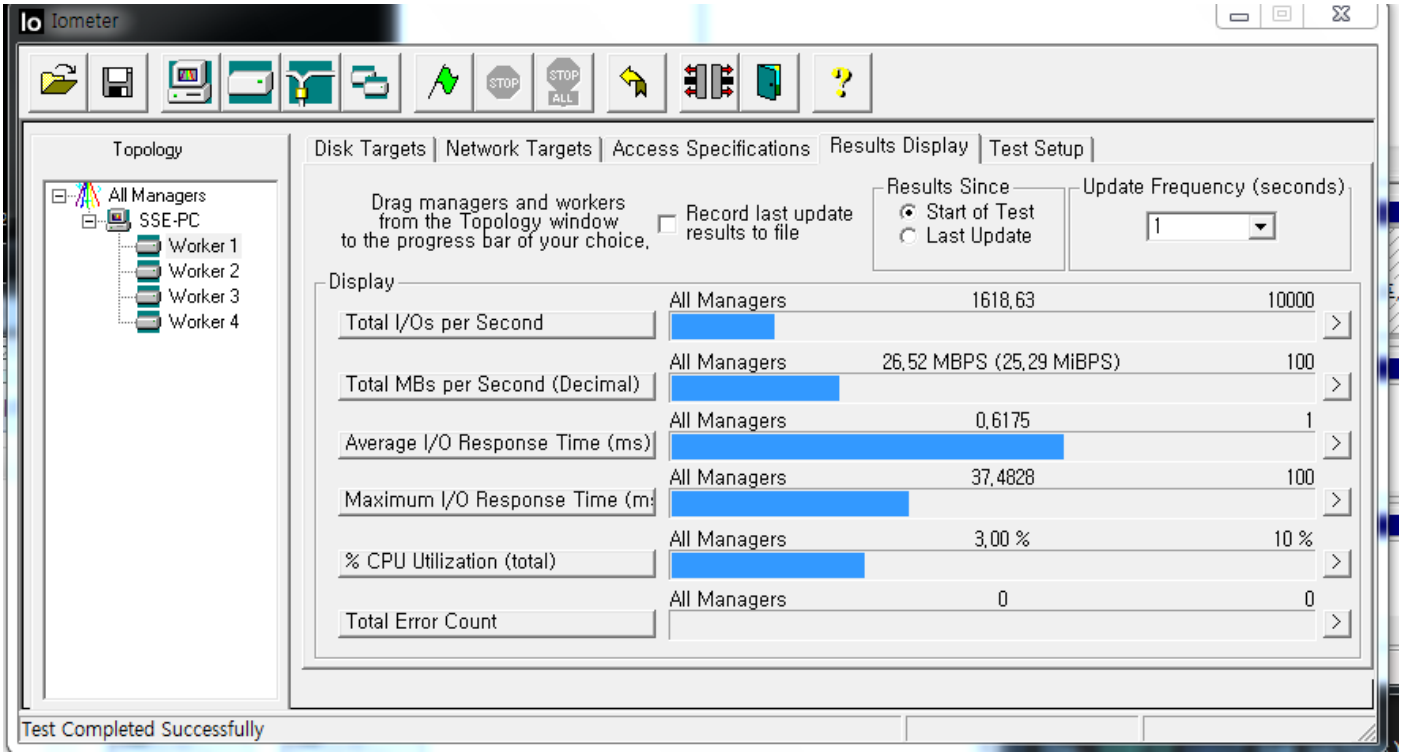
Sequential_read(16KB) with CMT_Ratio = 10



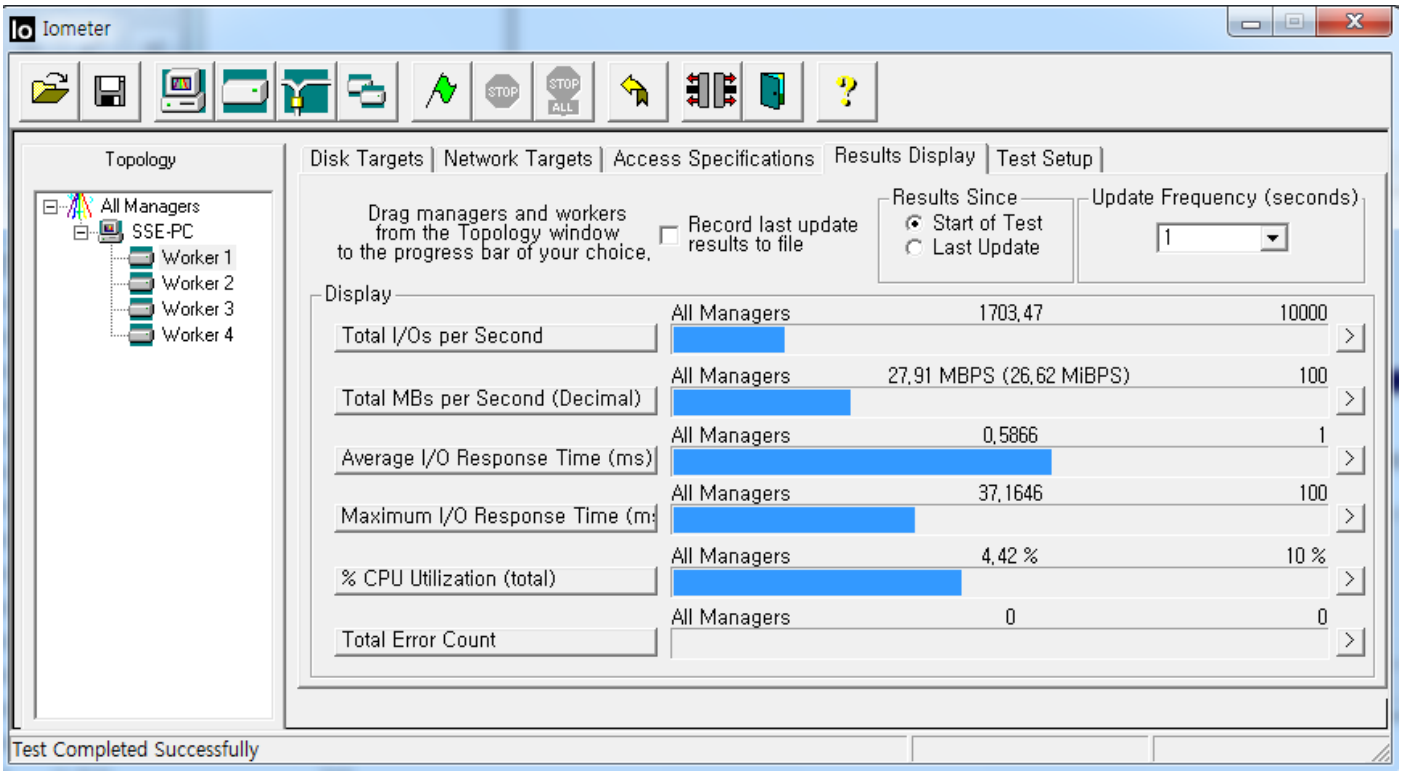
Sequential_read(16KB) with CMT_Ratio = 15



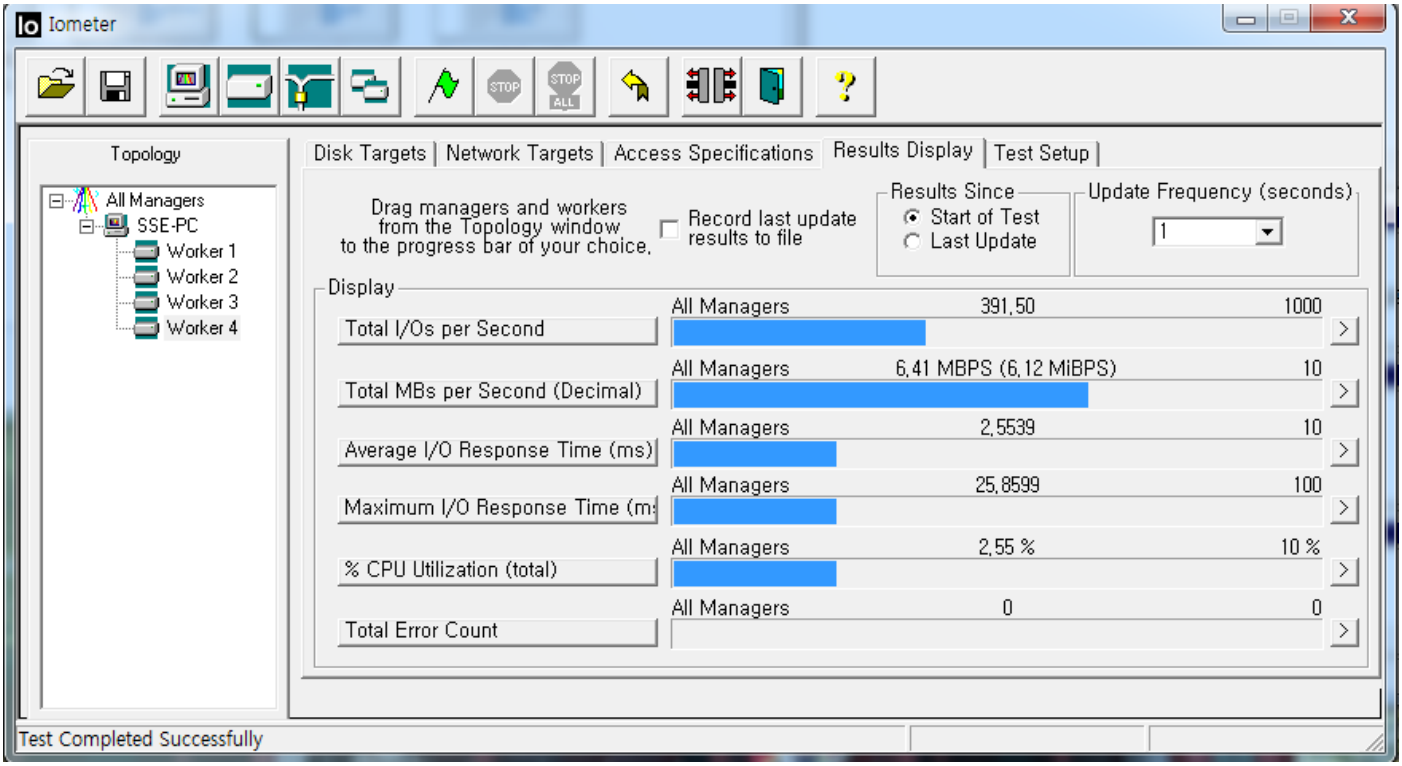
Sequential_read(16KB) with CMT_Ratio = 20



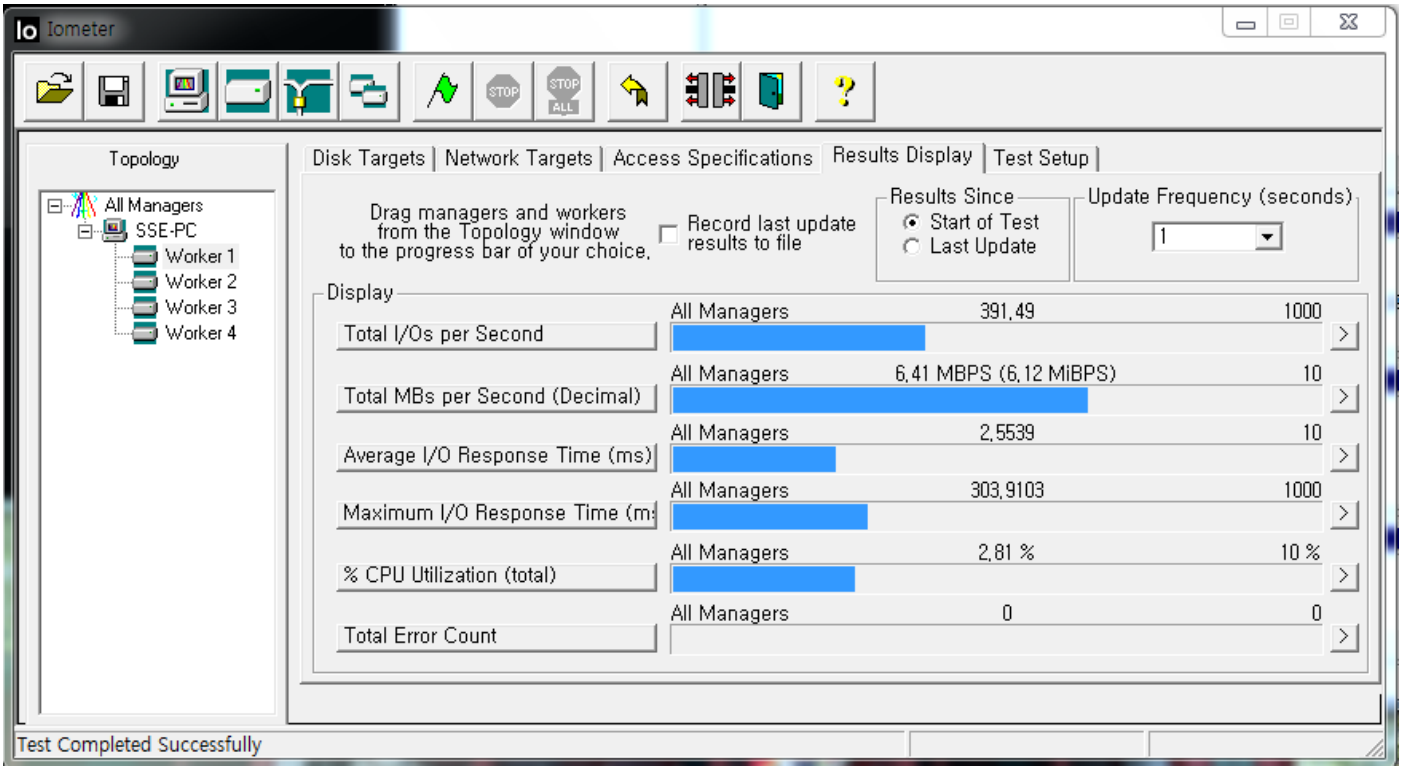
Sequential_read(16KB) with CMT_Ratio = 25



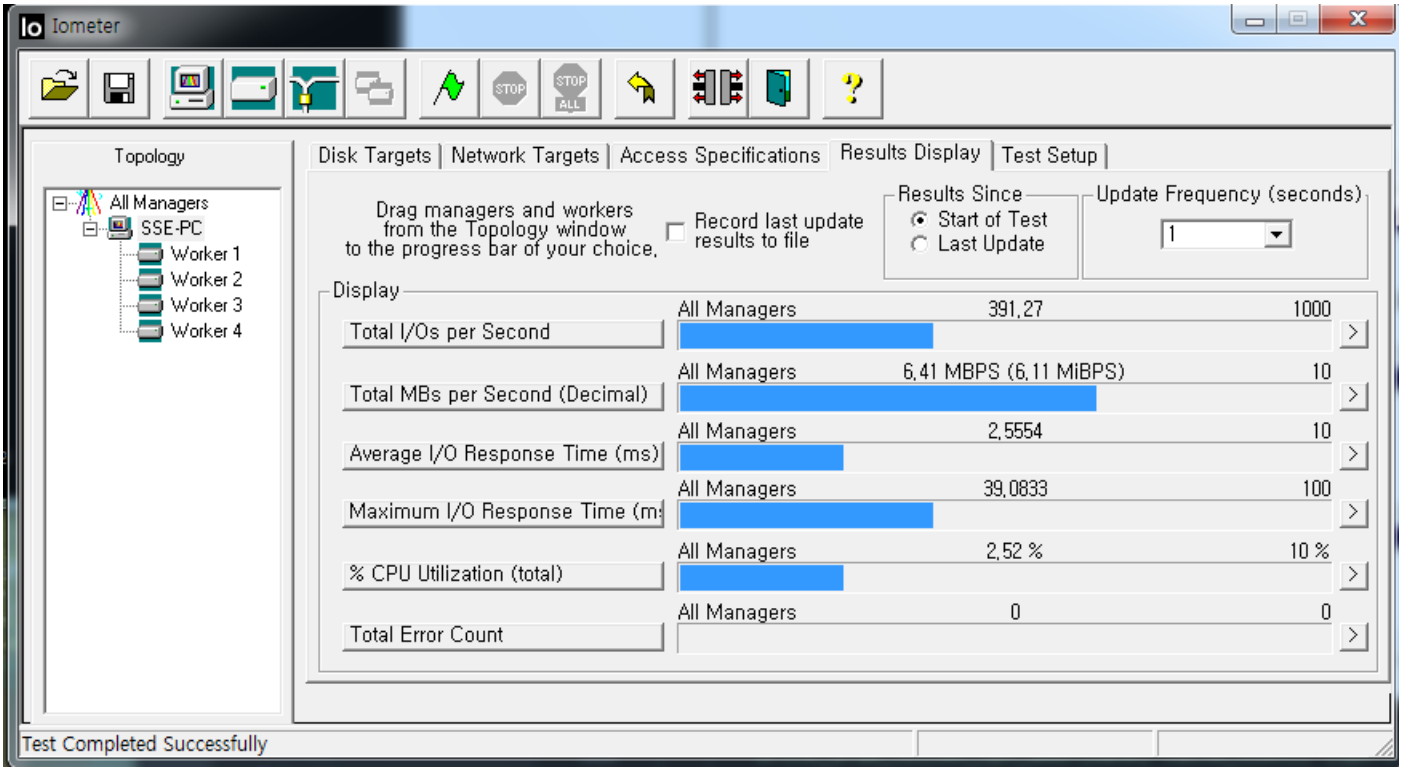
Sequential_write(16KB) with CMT_Ratio = 1



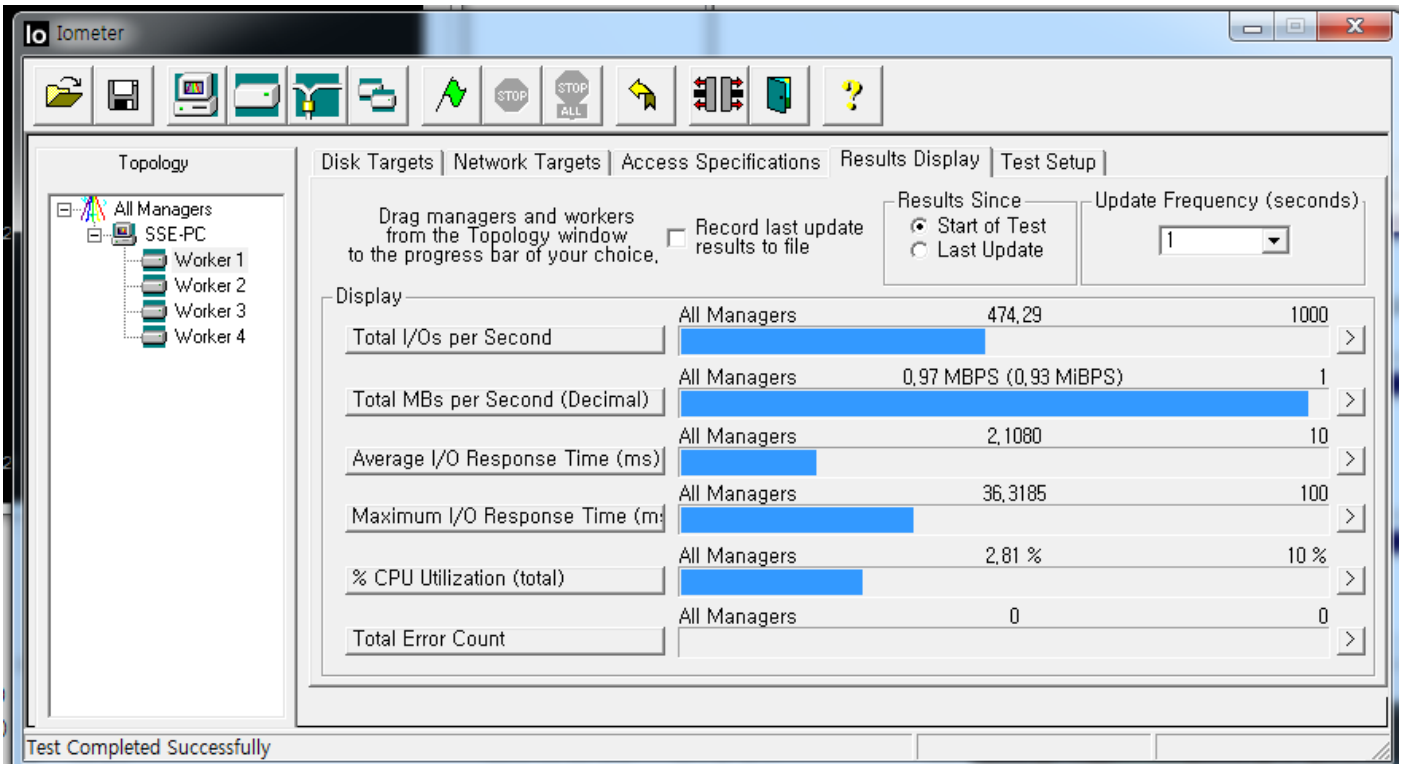
Sequential_write(16KB) with CMT_Ratio = 5



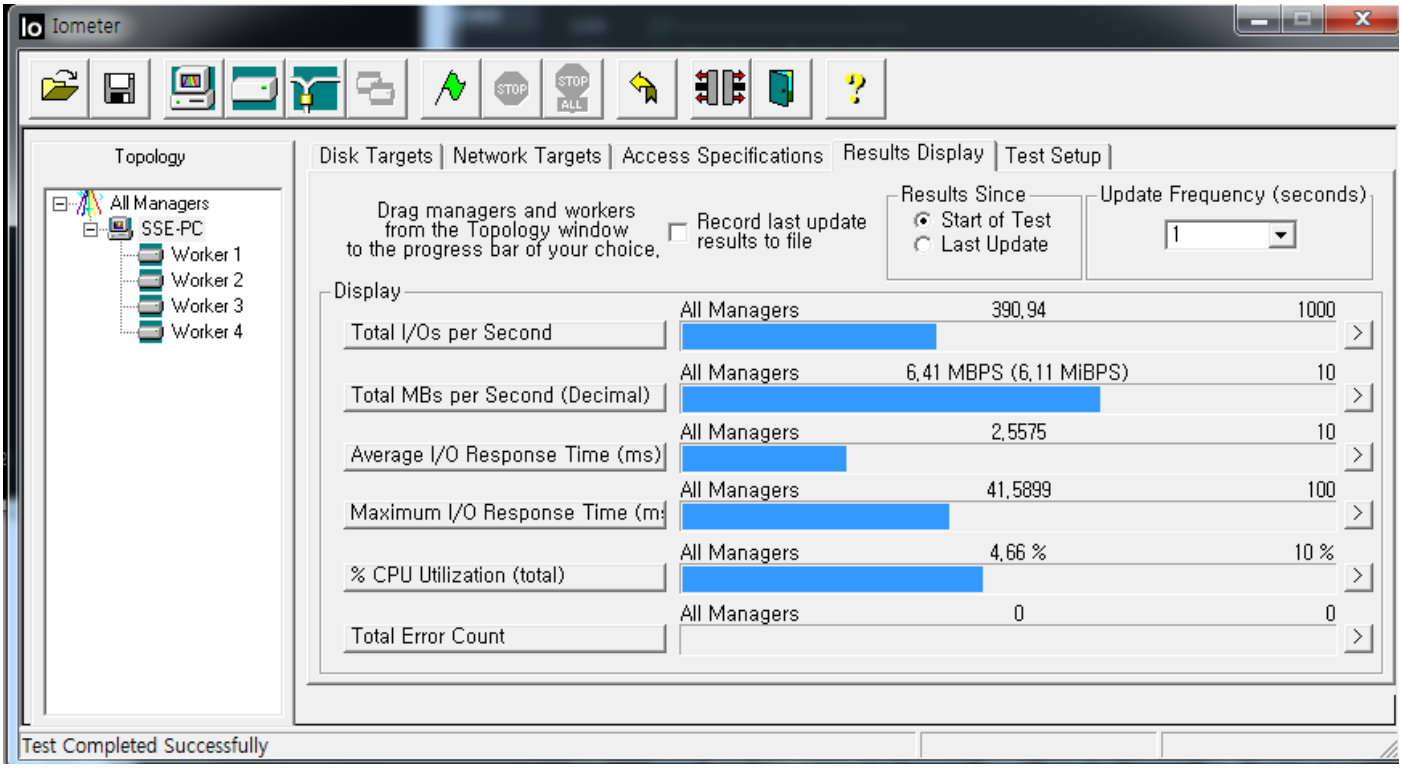
Sequential_write(16KB) with CMT_Ratio = 10



Sequential_write(16KB) with CMT_Ratio = 15



Sequential_write(16KB) with CMT_Ratio = 20



Sequential_write(16KB) with CMT_Ratio = 25