Intro. to Computer SW Systems Lab Report

[Cache Lab]

20220100 박기현

kihyun@postech.ac.kr

명예 서약 (Honor Code)

나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.

I completed this programming task without the improper help of others.

Part A. Writing a Cache Simulator

Cache 의 구조는 S(= 2^s)개의 Set 과 각 Set 당 E 개의 Line 이 존재하고, 각 Line 은 B(= 2^b) Bytes 의 크기를 갖는다. 또한, 각 Line 에는 Valid Bit 과 Tag 가 존재한다. 따라서 Cache Simulator 를 구현하기 위해 우선 1 개의 Cache Block 을 구현하는 struct 를 선언한다.

```
typedef struct //Cache Block
{
    int valid_bit;
    int tag;
    int* block;
    int LRU;
} Line;
```

여기서 LRU 는 Least Recently Used 로, evict 해야 하는 Line 을 선택하기 위한 Replacement Policy 를 구현하기 위한 값이다.

Line 을 1 개의 원소로 하는 S 개의 행과 E 개의 열을 가진 배열을 통해 Cache Memory 처럼 동작할 수 있도록 선언한다.

```
Line** Cache; //Cache Memory
```

이때 Cache Simulator 는 임의의 s(set index bits 의 수)와 E(set 당 lines 의 수), b(block bits 의 수)에 대해 올바르게 동작해야 하며, 이는 command-line arguments 로써 실행 이후에 입력 받으므로 동적할당을 이용한다.

getopt 함수를 이용하여 command-line arguments 를 parsing 할 수 있으므로, 이를 이용하여 각 arguments 를 입력 받는다.

```c
int opt = 0;
int help_flag = 0;
int verbose_flag = 0;
int s = 0; //Number of set index bits
int S = 0; //Number of sets
unsigned int E = 0; //Number of lines per set
int b = 0; //Number of block bits
unsigned int B = 0; //Block size
char* t = NULL; //trace file
```

```c
//Parse command-line arguments
while((opt = getopt(argc, argv, "hvs:E:b:t:")) != -1)
{
    switch(opt)
    {
        case 'h':
            help_flag = 1;
            break;
        case 'v':
            verbose_flag = 1;
            break;
        case 's':
            s = atoi(optarg);
            S = (unsigned)pow(2, s);
            break;
        case 'E':
            E = atoi(optarg);
            break;
        case 'b':
            b = atoi(optarg);
            B = (unsigned)pow(2, b);
            break;
        case 't':
            t = optarg;
            break;
    }
}
```

help_flag 의 여부에 따라 usage information 을 출력한다.

```c
void print_helpflag()
{
    if(help_flag)
    {
        printf("\nUsage: ./csim-ref [-hv] -s <s> -E <E> -b <b> -t <tracefile>\n");
        printf("  -h: Optional help flag that prints usage info\n");
        printf("  -v: Optional verbose flag that displays trace info\n");
        printf("  -s <s>: Number of set index bits (S = 2^s is the number of sets)\n");
        printf("  -E <E>: Associativity (number of lines per set)\n");
        printf("  -b <b>: Number of block bits (B = 2^b is the block size)\n");
        printf("  -t <tracefile>: Name of the valgrind trace to replay\n\n");
    }
    return;
}
```

입력 받은 arguments 를 이용하여 Cache Memory 의 크기를 동적할당한다.

```
void CacheInit()
{
    Cache = (Line**)malloc(sizeof(Line*) * S);
    for(int i = 0; i < S; i++)
    {
        Cache[i] = (Line*)malloc(sizeof(Line) * E);
        for(int j = 0; j < E; j++)
        {
            Cache[i][j].valid_bit = 0;
            Cache[i][j].tag = 0;
            Cache[i][j].block = (int*)malloc(sizeof(int) * B);
            Cache[i][j].LRU = 0;
        }
    }
    return;
}
```

Memory trace file 의 각 line 은 [space] operation address,size 로 이루어져 있다. operation 으로는 I, L, S, M 이 있으며, 각각 instruction load, data load, data store, data modify(data load + data store)를 뜻한다. 이때, 구현하고자 하는 Cache Simulator 는 data cache performance 에 대해서만 확인하므로 I operatoin 은 무시한다. command-line arguments 로 입력 받은 trace file 을 열고, 각 operation 에 해당하는 동작을 수행한다. Operation M 은 data modify(data load + data store)의 동작을 수행하므로 두 번의 접근이 필요하다.

```
void TraceInput()
{
    FILE* tracefile;

    char operation = NULL;
    unsigned long int address = 0;
    int size = 0;

    tracefile = fopen(t, "r");
    while(fscanf(tracefile, " %c %lx, %d", &operation, &address, &size) != EOF)
    {
        if(verbose_flag)
        {
            printf("%c %lx,%d", operation, address, size);
        }

        switch(operation)
        {
            case 'L':
                CacheSimulator(address);
                break;
            case 'M':
                CacheSimulator(address);
                CacheSimulator(address);
                break;
            case 'S':
                CacheSimulator(address);
                break;
        }

        if(verbose_flag)
        {
            printf("\n");
        }
    }
    fclose(tracefile);
    return;
}
```

가장 먼저 address 로부터 Tag, Set Index 를 확인한다. spatial locality 를 잘 활용하기 위해서 Middle-Bit Indexing(ttssbb 의 형태의 address)를 활용한다. 따라서 Tag 는 address 를 set index bits 와 block bits 만큼 right shift 하여 구할 수 있다. Set Index 는 block bits 만큼 right shift 한 후, 하위 set index bits 만큼 취하여 구할 수 있다.

```
unsigned long int tag = (address >> (s + b));
unsigned long int set = ((address >> b) & (S - 1));
```

그후 hit 여부 먼저 확인한다.

hit 여부를 확인하기 위해서는 같은 Set 에 있는 Line 중 Line 이 존재하고, Valid Bit 가 유효하며, Tag 가 일치하는지를 확인해야 한다. 모두 일치하는 Line 이 존재하면, hit_count 를 1 증가시키고, LRU 를 판단하기 위해 0 으로 초기화한다. 또한, 같은 Set 에 있는 유효한 Line 의 LRU 를 1 증가시킨다.

```
//Check hits
for(int i = 0; i < E; i++)
{
    if(Cache[set][i].valid_bit == 1)
    {
        if(Cache[set][i].tag == tag)
        {
            hit_or_not = 1;
            hit_line = i;
            Cache[set][i].LRU = 0;
            break;
        }
    }
}
if(hit_or_not == 1)
{
    for(int i = 0; i < E; i++)
    {
        if((i != hit_line) && (Cache[set][i].valid_bit == 1))
        {
            Cache[set][i].LRU++;
        }
    }

    hit_count++;
    if(verbose_flag)
    {
        printf(" hit");
    }
    return;
}
```

다음으로 miss 여부와 eviction 여부를 확인한다.

hit 이 되지 않았다면 무조건 miss 된 경우이므로 miss_count 를 1 증가시킨다. 같은 Set 에 유효하지 않은 Line 이 존재한다면 evict 할 필요가 없다. 반면, evict 해야 하는 경우에는 가장 LRU 값이 큰, 즉 가장 최근에 사용하지 않은 Line 을 evict 하고, eviction_count 를 1 증가시킨다.

Valid Bit 와 Tag, LRU 값을 업데이트하고 operation 에 대한 동작을 종료한다.

```
//Check misses and evictions
miss_count++;
if(verbose_flag)
{
    printf(" miss");
}

for(int i = 0; i < E; i++)
{
    if(Cache[set][i].valid_bit == 0)
    {
        evict_or_not = -1;
        eviction_line = i;
        break;
    }
    else
    {
        if(Cache[set][i].LRU > LRU_num)
        {
            LRU_num = Cache[set][i].LRU;
            evict_or_not = 1;
            eviction_line = i;
        }
    }
}
```

```
for(int i = 0; i < E; i++)
{
    if((i != eviction_line) && (Cache[set][i].valid_bit == 1))
    {
        Cache[set][i].LRU++;
    }
}

if(evict_or_not == -1)
{
    Cache[set][eviction_line].valid_bit = 1;
    Cache[set][eviction_line].tag = tag;
    Cache[set][eviction_line].LRU = 0;
}
else if(evict_or_not == 1)
{
    eviction_count++;
    if(verbose_flag)
    {
        printf(" eviction");
    }
    Cache[set][eviction_line].tag = tag;
    Cache[set][eviction_line].LRU = 0;
}
return;
```

trace file 의 모든 opeartion 에 대해 Cache Simulator 동작을 수행한 후, 동적할당을 해제하고 hit_count, miss_count, eviction_count 를 출력한 후 프로그램을 종료한다.

```
void DeleteCache()
{
    for(int i = 0; i < S; i++)
    {
        for(int j = 0; j < E; j++)
        {
            free(Cache[i][j].block);
        }
        free(Cache[i]);
    }
    free(Cache);
    return;
}
```

Cache Simulator 의 결과는 다음과 같다.

```
Part A: Testing cache simulator
Running ./test-csim
                        Your simulator     Reference simulator
Points (s,E,b)    Hits  Misses  Evicts    Hits  Misses  Evicts
     3 (1,1,1)       9       8       6       9       8       6  traces/yi2.trace
     3 (4,2,4)       4       5       2       4       5       2  traces/yi.trace
     3 (2,1,4)       2       3       1       2       3       1  traces/dave.trace
     3 (2,1,3)     167      71      67     167      71      67  traces/trans.trace
     3 (2,2,3)     201      37      29     201      37      29  traces/trans.trace
     3 (2,4,3)     212      26      10     212      26      10  traces/trans.trace
     3 (5,1,5)     231       7       0     231       7       0  traces/trans.trace
     6 (5,1,5)  265189   21775   21743  265189   21775   21743  traces/long.trace
    27
```

Part B. Optimizing Matrix Transpose

Cache Memory 의 s = 5, E = 1, b = 5 이므로, 총 $S = 2^5 = 32$ 개의 Set 이 존재하고, 각 Set 당 1 개의 Line 이 있으며, $B = 2^5 = 32$ Bytes 의 data 를 Cache 에 저장할 수 있다.

또한, 32 Bytes 마다 Set 이 1 증가하며, 1024 Bytes 마다 Tag 가 1 증가한다.

- 32 X 32 (M = 32, N = 32)

우선, trans 함수의 performance 를 확인해 보면, hits: 870, misses: 1183, evictions: 1151 임을 알 수 있다. 이때, 조금 더 구체적인 Cache 의 동작에 대해 알아보기 위해 Part A 에서 구현한 Cache Simulator 를 변형하여 활용한다.

```
L 603100,4 miss eviction    Set:  8, Tag: 180c
S 643100,4 miss eviction    Set:  8, Tag: 190c
L 603104,4 miss eviction    Set:  8, Tag: 180c
S 643180,4 miss             Set:  c, Tag: 190c
L 603108,4 hit              Set:  8, Tag: 180c
S 643200,4 miss             Set: 10, Tag: 190c
L 60310c,4 hit              Set:  8, Tag: 180c
S 643280,4 miss             Set: 14, Tag: 190c
L 603110,4 hit              Set:  8, Tag: 180c
S 643300,4 miss             Set: 18, Tag: 190c
L 603114,4 hit              Set:  8, Tag: 180c
S 643380,4 miss             Set: 1c, Tag: 190c
L 603118,4 hit              Set:  8, Tag: 180c
S 643400,4 miss             Set:  0, Tag: 190d
L 60311c,4 hit              Set:  8, Tag: 180c
S 643480,4 miss             Set:  4, Tag: 190d
L 603120,4 miss             Set:  9, Tag: 180c
S 643500,4 miss eviction    Set:  8, Tag: 190d
L 603124,4 hit              Set:  9, Tag: 180c
S 643580,4 miss eviction    Set:  c, Tag: 190d
L 603128,4 hit              Set:  9, Tag: 180c
S 643600,4 miss eviction    Set: 10, Tag: 190d
L 60312c,4 hit              Set:  9, Tag: 180c
S 643680,4 miss eviction    Set: 14, Tag: 190d
L 603130,4 hit              Set:  9, Tag: 180c
S 643700,4 miss eviction    Set: 18, Tag: 190d
L 603134,4 hit              Set:  9, Tag: 180c
```

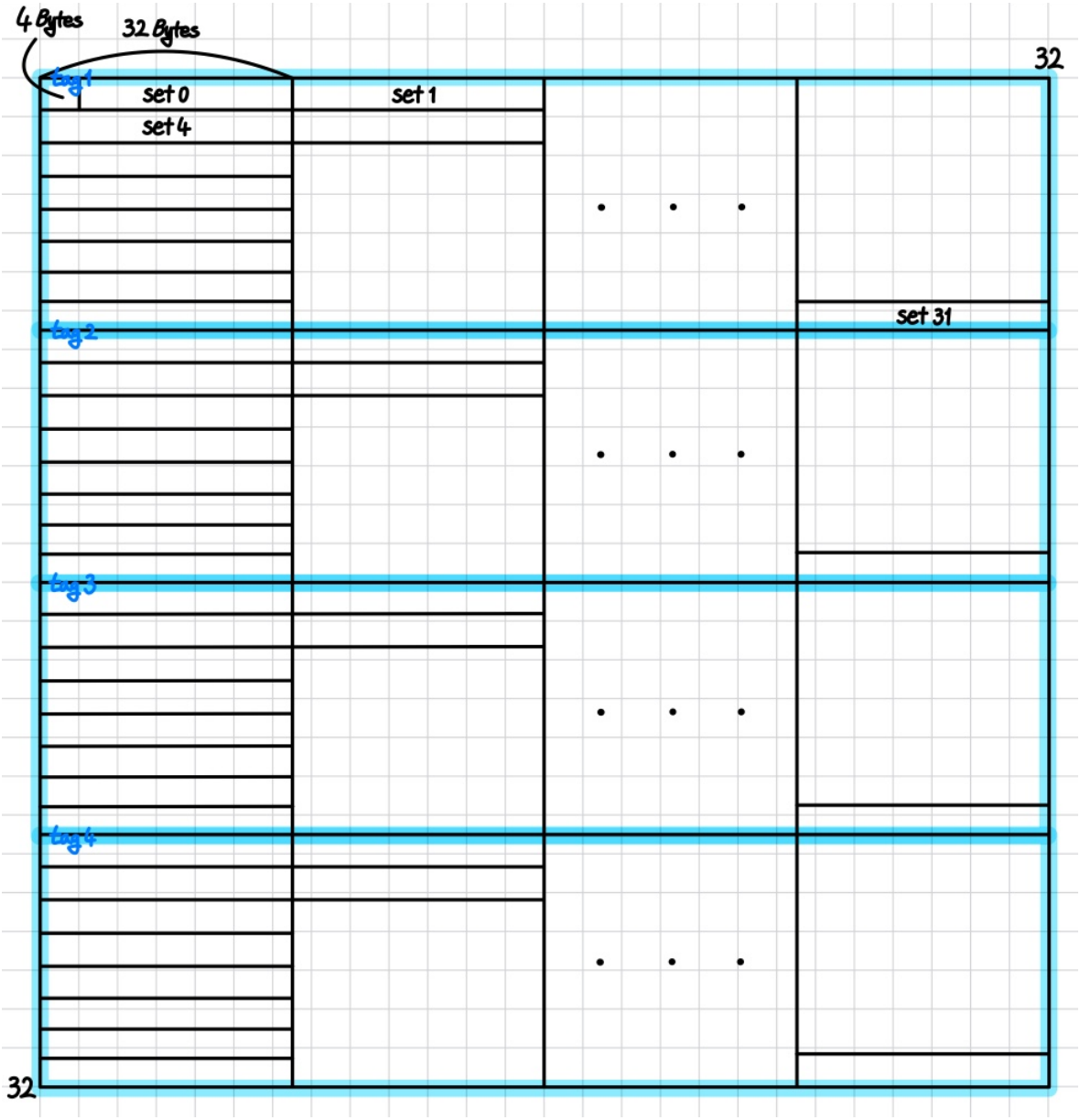
배열 A 를 transpose 하여 배열 B 에 저장하는 함수이므로, operation L 에 해당하는 address 는 배열 A, operation S 에 해당하는 address 는 배열 B 임을 알 수 있다.

배열의 data type 이 int 이므로 각 원소는 4 Bytes 이고, 따라서 8 개의 원소까지 Cache 에 저장 가능하다.

또한, 위의 operation S 에 해당하는 Set 과 Tag 를 보면 8 개 line 의 Set 과 Tag 가 동일하고, 그 이후부터 Set 이 1 증가하는 것을 알 수 있다.

이를 이용하여 배열 A 의 Row-major 원소 8 개씩 local variable 에 저장하고, 이를 배열 B 에 저장한다.

나아가 배열 B 에 저장할 때 8 개의 각각 Set 이 다른 address 에 해당하는 원소에 접근하게 되므로, spatial locality 를 활용하여 8 X 8 블록으로 나누어 반복문을 수행한다.



```
if(M == 32 && N == 32)
{
    for(int i = 0; i < N; i += 8)
    {
        for(int j = 0; j < M; j += 8)
        {
            for(int k = i; k < i + 8; k++)
            {
                temp1 = A[k][j];
                temp2 = A[k][j + 1];
                temp3 = A[k][j + 2];
                temp4 = A[k][j + 3];
                temp5 = A[k][j + 4];
                temp6 = A[k][j + 5];
                temp7 = A[k][j + 6];
                temp8 = A[k][j + 7];

                B[j][k] = temp1;
                B[j + 1][k] = temp2;
                B[j + 2][k] = temp3;
                B[j + 3][k] = temp4;
                B[j + 4][k] = temp5;
                B[j + 5][k] = temp6;
                B[j + 6][k] = temp7;
                B[j + 7][k] = temp8;
            }
        }
    }
}
```

32 X 32 Matrix Transpose 의 결과는 다음과 같다.

```
[kihyun@programming2 cachelab-handout]$ ./test-trans -M 32 -N 32

Function 0 (2 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 0 (Transpose submission): hits:1766, misses:287, evictions:255

Function 1 (2 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 1 (Simple row-wise scan transpose): hits:870, misses:1183, evictions:1151

Summary for official submission (func 0): correctness=1 misses=287

TEST_TRANS_RESULTS=1:287
```

Cache Simulator 를 통해서도 첫 열의 블록에 저장할 때를 제외하고 대부분 hit 이 일어나는 것을 확인할 수 있었다.

```
L 603100,4 miss eviction    Set:  8, Tag: 180c
L 603104,4 hit              Set:  8, Tag: 180c
L 603108,4 hit              Set:  8, Tag: 180c
L 60310c,4 hit              Set:  8, Tag: 180c
L 603110,4 hit              Set:  8, Tag: 180c
L 603114,4 hit              Set:  8, Tag: 180c
L 603118,4 hit              Set:  8, Tag: 180c
L 60311c,4 hit              Set:  8, Tag: 180c
S 643100,4 miss eviction    Set:  8, Tag: 190c
S 643180,4 miss             Set:  c, Tag: 190c
S 643200,4 miss             Set: 10, Tag: 190c
S 643280,4 miss             Set: 14, Tag: 190c
S 643300,4 miss             Set: 18, Tag: 190c
S 643380,4 miss             Set: 1c, Tag: 190c
S 643400,4 miss             Set:  0, Tag: 190d
S 643480,4 miss             Set:  4, Tag: 190d
L 603180,4 miss eviction    Set:  c, Tag: 180c
L 603184,4 hit              Set:  c, Tag: 180c
L 603188,4 hit              Set:  c, Tag: 180c
L 60318c,4 hit              Set:  c, Tag: 180c
L 603190,4 hit              Set:  c, Tag: 180c
L 603194,4 hit              Set:  c, Tag: 180c
L 603198,4 hit              Set:  c, Tag: 180c
L 60319c,4 hit              Set:  c, Tag: 180c
S 643104,4 hit              Set:  8, Tag: 190c
S 643184,4 miss eviction    Set:  c, Tag: 190c
S 643204,4 hit              Set: 10, Tag: 190c
```

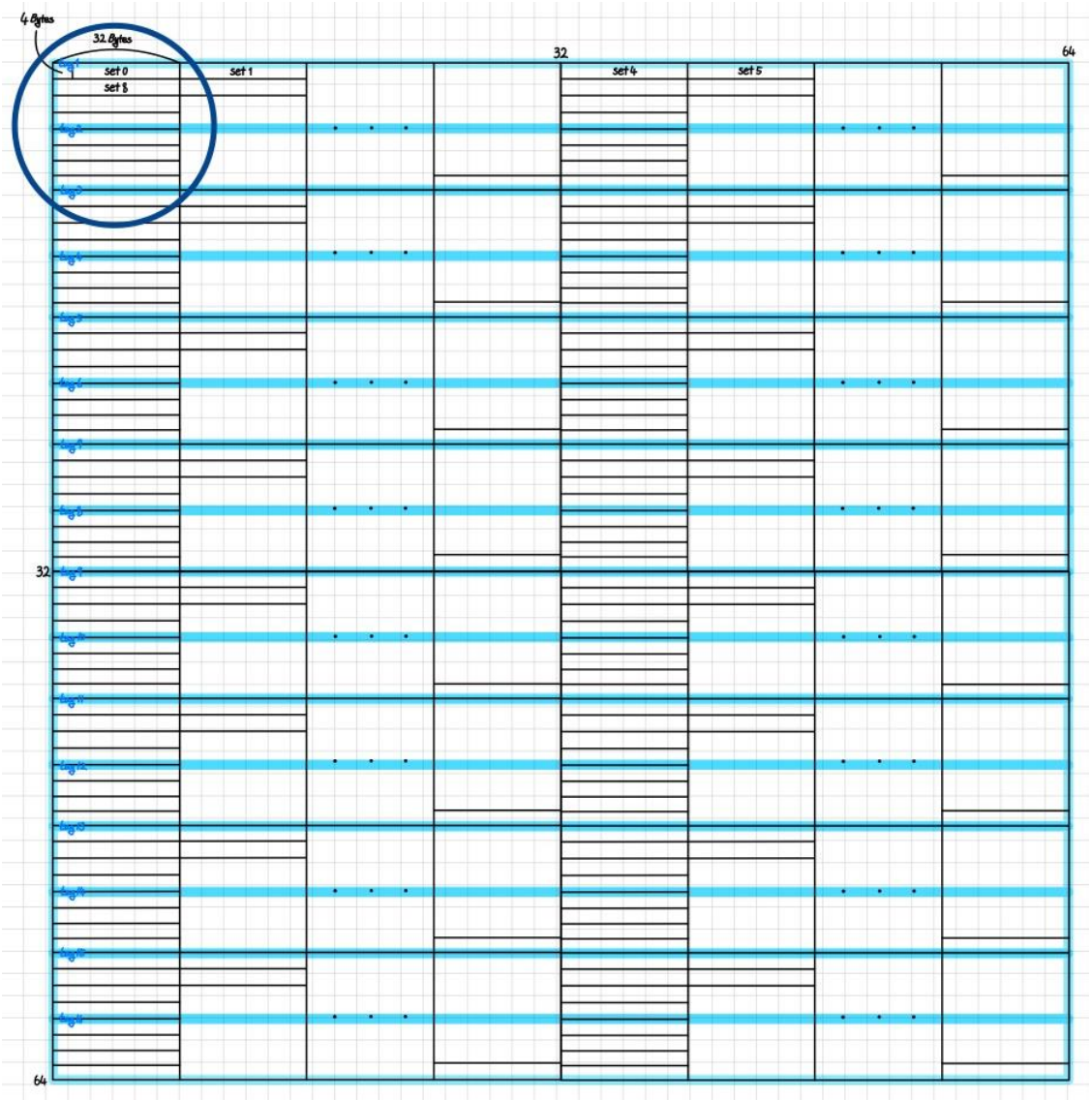
```
L 603140,4 miss             Set:  a, Tag: 180c
L 603144,4 hit              Set:  a, Tag: 180c
L 603148,4 hit              Set:  a, Tag: 180c
L 60314c,4 hit              Set:  a, Tag: 180c
L 603150,4 hit              Set:  a, Tag: 180c
L 603154,4 hit              Set:  a, Tag: 180c
L 603158,4 hit              Set:  a, Tag: 180c
L 60315c,4 hit              Set:  a, Tag: 180c
S 643900,4 miss eviction    Set:  8, Tag: 190e
S 643980,4 miss eviction    Set:  c, Tag: 190e
S 643a00,4 miss eviction    Set: 10, Tag: 190e
S 643a80,4 miss eviction    Set: 14, Tag: 190e
S 643b00,4 miss eviction    Set: 18, Tag: 190e
S 643b80,4 miss eviction    Set: 1c, Tag: 190e
S 643c00,4 miss eviction    Set:  0, Tag: 190f
S 643c80,4 miss eviction    Set:  4, Tag: 190f
L 6031c0,4 miss             Set:  e, Tag: 180c
L 6031c4,4 hit              Set:  e, Tag: 180c
L 6031c8,4 hit              Set:  e, Tag: 180c
L 6031cc,4 hit              Set:  e, Tag: 180c
L 6031d0,4 hit              Set:  e, Tag: 180c
L 6031d4,4 hit              Set:  e, Tag: 180c
L 6031d8,4 hit              Set:  e, Tag: 180c
L 6031dc,4 hit              Set:  e, Tag: 180c
S 643904,4 hit              Set:  8, Tag: 190e
S 643984,4 hit              Set:  c, Tag: 190e
S 643a04,4 hit              Set: 10, Tag: 190e
S 643a84,4 hit              Set: 14, Tag: 190e
S 643b04,4 hit              Set: 18, Tag: 190e
S 643b84,4 hit              Set: 1c, Tag: 190e
```

- 64 X 64 (M = 64, N = 64)

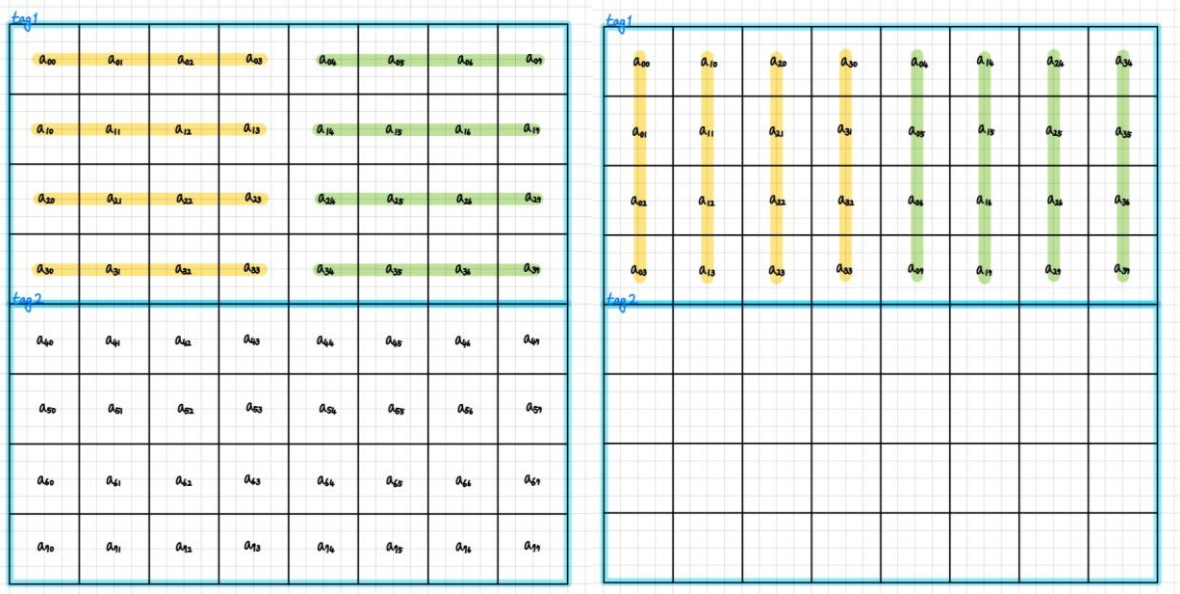
32 X 32 Matrix Transpose 에서 구현한 대로 적용하여 Cache Simulator 를 통해 확인해 보면 다음과 같다.

```
L 603100,4 miss eviction    Set:  8, Tag: 180c
L 603104,4 hit              Set:  8, Tag: 180c
L 603108,4 hit              Set:  8, Tag: 180c
L 60310c,4 hit              Set:  8, Tag: 180c
L 603110,4 hit              Set:  8, Tag: 180c
L 603114,4 hit              Set:  8, Tag: 180c
L 603118,4 hit              Set:  8, Tag: 180c
L 60311c,4 hit              Set:  8, Tag: 180c
S 643100,4 miss eviction    Set:  8, Tag: 190c
S 643200,4 miss             Set: 10, Tag: 190c
S 643300,4 miss             Set: 18, Tag: 190c
S 643400,4 miss             Set:  0, Tag: 190d
S 643500,4 miss eviction    Set:  8, Tag: 190d
S 643600,4 miss eviction    Set: 10, Tag: 190d
S 643700,4 miss eviction    Set: 18, Tag: 190d
S 643800,4 miss eviction    Set:  0, Tag: 190e
L 603200,4 miss eviction    Set: 10, Tag: 180c
L 603204,4 hit              Set: 10, Tag: 180c
L 603208,4 hit              Set: 10, Tag: 180c
L 60320c,4 hit              Set: 10, Tag: 180c
L 603210,4 hit              Set: 10, Tag: 180c
L 603214,4 hit              Set: 10, Tag: 180c
L 603218,4 hit              Set: 10, Tag: 180c
L 60321c,4 hit              Set: 10, Tag: 180c
S 643104,4 miss eviction    Set:  8, Tag: 190c
S 643204,4 miss eviction    Set: 10, Tag: 190c
S 643304,4 miss eviction    Set: 18, Tag: 190c
S 643404,4 miss eviction    Set:  0, Tag: 190d
```

이때, 32 X 32 Matrix 의 경우와 달리, 64 X 64 Matrix 는 8 X 8 블록 내에서 Tag 가 바뀌므로 hit 이 아닌 miss eviction 이 발생한다. 따라서 이를 해결하기 위해 8 X 8 블록 내에서도 다시 4 X 8 블록 2 개로 나누어 miss eviction 의 수를 줄일 수 있다.



우선, 위 4 X 8 블록에서 배열 A 의 원소를 local variable 에 저장하는 과정은 똑같이 수행한다. 배열 B 에 저장할 때 기존에는 행과 열을 바꾼 값에 바로 저장했다면, Tag 가 달라지는 것을 해결하기 위해 4 X 4 블록 내에서 transpose 를 수행한다.



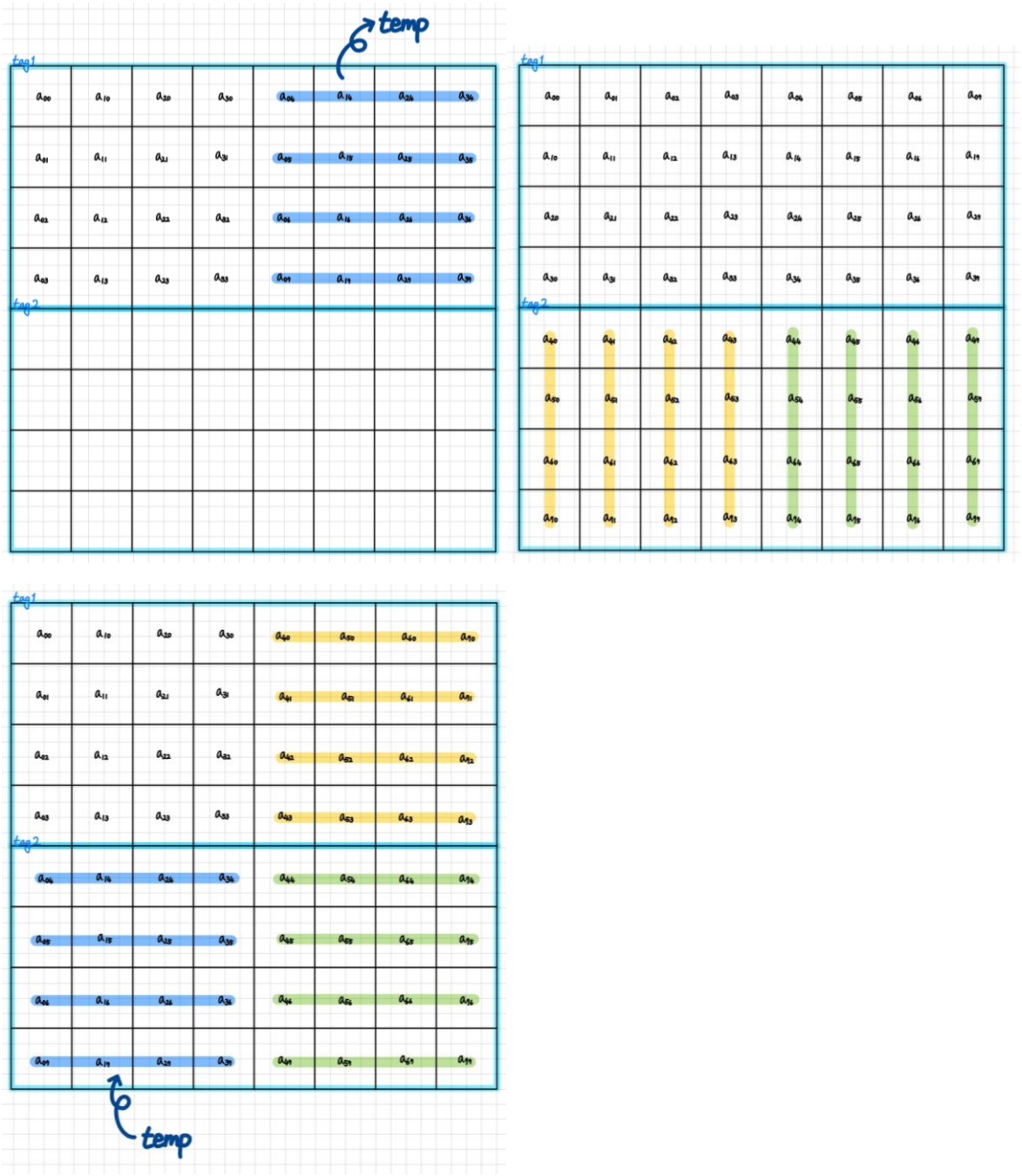

```
for(int i = 0; i < N; i += 8)
{
    for(int j = 0; j < M; j += 8)
    {
        for(int k = i; k < i + 4; k++)
        {
            temp1 = A[k][j];
            temp2 = A[k][j + 1];
            temp3 = A[k][j + 2];
            temp4 = A[k][j + 3];
            temp5 = A[k][j + 4];
            temp6 = A[k][j + 5];
            temp7 = A[k][j + 6];
            temp8 = A[k][j + 7];

            B[j][k] = temp1;
            B[j + 1][k] = temp2;
            B[j + 2][k] = temp3;
            B[j + 3][k] = temp4;
            B[j][k + 4] = temp5;
            B[j + 1][k + 4] = temp6;
            B[j + 2][k + 4] = temp7;
            B[j + 3][k + 4] = temp8;
        }
```

다음으로 아래 4 X 8 블록에서 역시 배열 A 의 원소를 local variable 에 저장하는 과정은 똑같이 수행한다.

그후 오른쪽 위 4 X 4 블록의 값을 왼쪽 아래 4 X 4 블록에 옮기는 과정을 수행한다. 이때, spatial locality 를 활용하기 위해 Row-major 원소 4 개씩 먼저 local variable 에 저장하여 옮긴 후, 바로 올바른 값을 저장한다.

그리고 이어서 배열 B 의 남은 위치에 Row-major 로 저장하면 miss 수를 줄일 수 있다.



```
for(int l = 0; l < 4; l++)
{
    temp1 = A[i + 4][j + l];
    temp2 = A[i + 5][j + l];
    temp3 = A[i + 6][j + l];
    temp4 = A[i + 7][j + l];
    temp5 = A[i + 4][j + 4 + l];
    temp6 = A[i + 5][j + 4 + l];
    temp7 = A[i + 6][j + 4 + l];
    temp8 = A[i + 7][j + 4 + l];

    temp9 = B[j + l][i + 4];
    temp10 = B[j + l][i + 5];
    temp11 = B[j + l][i + 6];
    temp12 = B[j + l][i + 7];

    B[j + l][i + 4] = temp1;
    B[j + l][i + 5] = temp2;
    B[j + l][i + 6] = temp3;
    B[j + l][i + 7] = temp4;

    B[j + 4 + l][i] = temp9;
    B[j + 4 + l][i + 1] = temp10;
    B[j + 4 + l][i + 2] = temp11;
    B[j + 4 + l][i + 3] = temp12;

    B[j + 4 + l][i + 4] = temp5;
    B[j + 4 + l][i + 5] = temp6;
    B[j + 4 + l][i + 6] = temp7;
    B[j + 4 + l][i + 7] = temp8;
}
```

64 X 64 Matrix Transpose 의 결과는 다음과 같다.

```
[kihyun@programming2 cachelab-handout]$ ./test-trans -M 64 -N 64

Function 0 (2 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 0 (Transpose submission): hits:9138, misses:1107, evictions:1075

Function 1 (2 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 1 (Simple row-wise scan transpose): hits:3474, misses:4723, evictions:4691

Summary for official submission (func 0): correctness=1 misses=1107

TEST TRANS RESULTS=1:1107
```

Cache Simulator 를 통해서도 아래 4 X 8 블록에서 배열 A 의 값을 불러올 때를 제외하고 대부분 hit 이 일어나는 것을 확인할 수 있었다.

```
L 603100,4 miss eviction    Set:  8, Tag: 180c
L 603104,4 hit              Set:  8, Tag: 180c
L 603108,4 hit              Set:  8, Tag: 180c
L 60310c,4 hit              Set:  8, Tag: 180c
L 603110,4 hit              Set:  8, Tag: 180c
L 603114,4 hit              Set:  8, Tag: 180c
L 603118,4 hit              Set:  8, Tag: 180c
L 60311c,4 hit              Set:  8, Tag: 180c
S 643100,4 miss eviction    Set:  8, Tag: 190c
S 643200,4 miss             Set: 10, Tag: 190c
S 643300,4 miss             Set: 18, Tag: 190c
S 643400,4 miss             Set:  0, Tag: 190d
S 643110,4 hit              Set:  8, Tag: 190c
S 643210,4 hit              Set: 10, Tag: 190c
S 643310,4 hit              Set: 18, Tag: 190c
S 643410,4 hit              Set:  0, Tag: 190d
L 603200,4 miss eviction    Set: 10, Tag: 180c
L 603204,4 hit              Set: 10, Tag: 180c
L 603208,4 hit              Set: 10, Tag: 180c
L 60320c,4 hit              Set: 10, Tag: 180c
L 603210,4 hit              Set: 10, Tag: 180c
L 603214,4 hit              Set: 10, Tag: 180c
L 603218,4 hit              Set: 10, Tag: 180c
L 60321c,4 hit              Set: 10, Tag: 180c
S 643104,4 hit              Set:  8, Tag: 190c
S 643204,4 miss eviction    Set: 10, Tag: 190c
S 643304,4 hit              Set: 18, Tag: 190c
S 643404,4 hit              Set:  0, Tag: 190d
```

```
L 603500,4 miss eviction    Set:  8, Tag: 180d
L 603600,4 miss eviction    Set: 10, Tag: 180d
L 603700,4 miss eviction    Set: 18, Tag: 180d
L 603800,4 miss eviction    Set:  0, Tag: 180e
L 603510,4 hit              Set:  8, Tag: 180d
L 603610,4 hit              Set: 10, Tag: 180d
L 603710,4 hit              Set: 18, Tag: 180d
L 603810,4 hit              Set:  0, Tag: 180e
L 643110,4 miss eviction    Set:  8, Tag: 190c
L 643114,4 hit              Set:  8, Tag: 190c
L 643118,4 hit              Set:  8, Tag: 190c
L 64311c,4 hit              Set:  8, Tag: 190c
S 643110,4 hit              Set:  8, Tag: 190c
S 643114,4 hit              Set:  8, Tag: 190c
S 643118,4 hit              Set:  8, Tag: 190c
S 64311c,4 hit              Set:  8, Tag: 190c
S 643500,4 miss eviction    Set:  8, Tag: 190d
S 643504,4 hit              Set:  8, Tag: 190d
S 643508,4 hit              Set:  8, Tag: 190d
S 64350c,4 hit              Set:  8, Tag: 190d
S 643510,4 hit              Set:  8, Tag: 190d
S 643514,4 hit              Set:  8, Tag: 190d
S 643518,4 hit              Set:  8, Tag: 190d
```

- 61 X 67 (M = 61, N = 67)

정방행렬이 아니며, M 과 N 모두 8 의 배수도 아니므로 위의 두 경우처럼 Set 과 Tag 를 기준으로 일정하게 블록을 나눌 수 없다. 따라서 32 X 32 Matrix Transpose 에서 사용한 알고리즘을 활용하여 64 <= i < 67, 56 <= j < 61 인 경우에서의 조건만 추가하였다.

```
else if(M == 61 && N == 67)
{
    for(int i = 0; i < N; i += 8)
    {
        for(int j = 0; j < M; j += 8)
        {
            for(int k = i; (k < i + 8) && (k < N); k++)
            {
                temp1 = A[k][j];
                temp2 = A[k][j + 1];
                temp3 = A[k][j + 2];
                temp4 = A[k][j + 3];
                temp5 = A[k][j + 4];
                if(j != 56)
                {
                    temp6 = A[k][j + 5];
                    temp7 = A[k][j + 6];
                    temp8 = A[k][j + 7];
                }

                B[j][k] = temp1;
                B[j + 1][k] = temp2;
                B[j + 2][k] = temp3;
                B[j + 3][k] = temp4;
                B[j + 4][k] = temp5;
                if(j != 56)
                {
                    B[j + 5][k] = temp6;
                    B[j + 6][k] = temp7;
                    B[j + 7][k] = temp8;
                }
            }
        }
    }
}
```

61 X 67 Matrix Transpose 의 결과는 다음과 같다.

```
[kihyun@programming2 cachelab-handout]$ ./test-trans -M 61 -N 67

Function 0 (2 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 0 (Transpose submission): hits:6182, misses:1997, evictions:1965

Function 1 (2 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 1 (Simple row-wise scan transpose): hits:3756, misses:4423, evictions:4391

Summary for official submission (func 0): correctness=1 misses=1997

TEST_TRANS_RESULTS=1:1997
```

```
[kihyun@programming2 cachelab-handout]$ python2 driver.py
Part A: Testing cache simulator
Running ./test-csim
                        Your simulator     Reference simulator
Points (s,E,b)    Hits  Misses  Evicts    Hits  Misses  Evicts
     3 (1,1,1)       9       8       6       9       8       6  traces/yi2.trace
     3 (4,2,4)       4       5       2       4       5       2  traces/yi.trace
     3 (2,1,4)       2       3       1       2       3       1  traces/dave.trace
     3 (2,1,3)     167      71      67     167      71      67  traces/trans.trace
     3 (2,2,3)     201      37      29     201      37      29  traces/trans.trace
     3 (2,4,3)     212      26      10     212      26      10  traces/trans.trace
     3 (5,1,5)     231       7       0     231       7       0  traces/trans.trace
     6 (5,1,5)  265189   21775   21743  265189   21775   21743  traces/long.trace
    27


Part B: Testing transpose function
Running ./test-trans -M 32 -N 32
Running ./test-trans -M 64 -N 64
Running ./test-trans -M 61 -N 67

Cache Lab summary:
                        Points   Max pts      Misses
Csim correctness          27.0        27
Trans perf 32x32           8.0         8         287
Trans perf 64x64           8.0         8        1107
Trans perf 61x67          10.0        10        1997
          Total points    53.0        53
```