

# 하드웨어 시스템 설계 2주차 실습 보고서

2017-12751 컴퓨터공학부 이동학

Goal: Implement matrix-vector multiplication in C++ by splitting the matrix into tiles

Code: src/fpga\_api.cpp

```
void FPGA::largeMV(const float* large_mat, const float* input, float* output, int num_input, int num_output)
{
    float* vec = this->vector();
    float* mat = this->matrix();

    // 0) Initialize output vector
    for(int i = 0; i < num_output; ++i)
    {
        output[i] = 0;
    }

    for(int i = 0; i < num_output; i += m_size_)
    {
        for(int j = 0; j < num_input; j += v_size_)
        {
            // 0) Initialize input vector
            int block_row = min(m_size_, num_output-i);
            int block_col = min(v_size_, num_input-j);

            // 1) Assign a vector
            memset(vec, 0, v_size_*sizeof(float));
            memcpy(vec, input + j, block_col*sizeof(float));

            // 2) Assign a matrix
            memset(mat, 0, m_size_*v_size_*sizeof(float));

            for(int k = 0; k < block_row; k++) {
                memcpy(mat + block_col*k, large_mat + num_input*(i+k) + j, block_col*sizeof(float));
            }

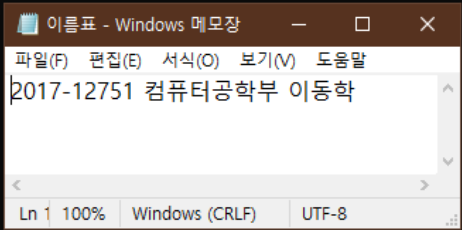
            // 3) Call a function `block_call()` to execute MV multiplication
            const float* ret = this->blockMV();

            // 4) Accumulate intermediate results
            for(int row = 0; row < block_row; ++row)
            {
                output[i + row] += ret[row];
            }
        }
    }
}
```

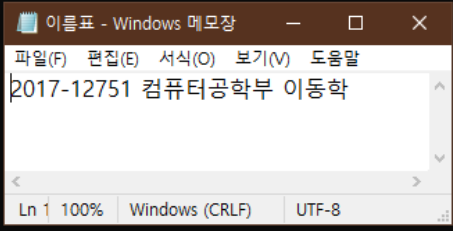
Explain: large\_mat 행렬과 input 벡터의 곱을 m\*v 크기의 행렬과 v 크기의 벡터의 곱으로 나누어 계산한 후 다시 합치는 코드이다. for문을 한바퀴 돌 때마다 input 벡터의 알맞은 위치로부터 block\_col 개의 값을 vec 벡터에 복사한 후, large\_mat 행렬의 알맞은 위치로부터 block\_col 개의 값을 mat 행렬의 알맞은 위치에 복사하는 과정을 block\_row 번 반복합니다. 복사가 완료된 vec 벡터와 mat 행렬을 곱한 중간 결과를 output에 저장합니다.

Result:

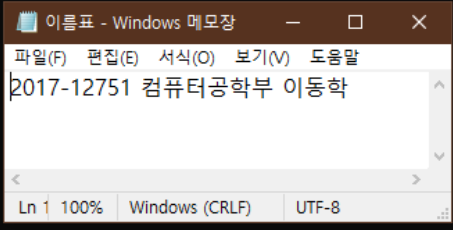
```
root@a579e75b7786:~# python eval.py
read dataset...
('images', (10000, 28, 28))
create network...
run test...
{'accuracy': 0.918,
 'avg_num_call': 627,
 'm_size': 64,
 'total_image': 10000,
 'total_time': 31.45689821243286,
 'v_size': 64}
root@a579e75b7786:~#
```



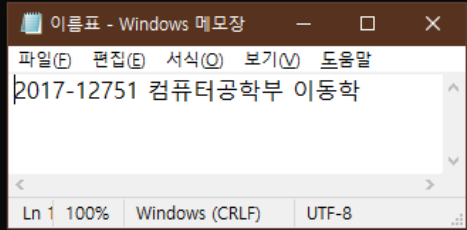
```
root@a579e75b7786:~# root@a579e75b7786:~# python eval.py
read dataset...
('images', (10000, 28, 28))
create network...
run test...
{'accuracy': 0.9159,
 'avg_num_call': 9375,
 'm_size': 16,
 'total_image': 10000,
 'total_time': 29.635473012924194,
 'v_size': 16}
root@a579e75b7786:~#
```



```
root@a579e75b7786:~# root@a579e75b7786:~# python eval.py
read dataset...
('images', (10000, 28, 28))
create network...
run test...
{'accuracy': 0.9159,
 'avg_num_call': 18750,
 'm_size': 8,
 'total_image': 10000,
 'total_time': 33.11879777908325,
 'v_size': 16}
root@a579e75b7786:~#
```



```
root@a579e75b7786:~# root@a579e75b7786:~# python eval.py
read dataset...
('images', (10000, 28, 28))
create network...
run test...
{'accuracy': 0.9159,
 'avg_num_call': 18750,
 'm_size': 16,
 'total_image': 10000,
 'total_time': 39.3908588886261,
 'v_size': 8}
root@a579e75b7786:~#
```



Discussion: 블록 크기를 64\*64, 16\*16, 8\*16, 16\*8의 4가지로 실행해본 결과, avg\_num\_call의 크기는 블록 크기에 반비례 합니다. 정확도에 유의미한 차이는 보이지 않으며, 블록 크기가 같은데도 16\*8의 total\_time이 8\*16의 total\_time보다 더 큰 이유는 mat을 복사하는 과정에서 for문의 반복 횟수가 더 많기 때문으로 생각됩니다.