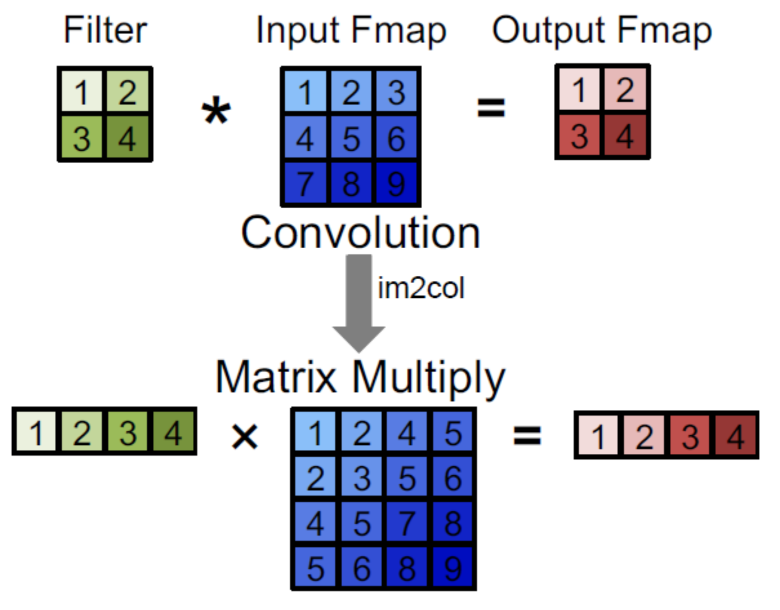
Mapping and Accelerating a Convolutional Neural Network on a GPU

Haung Shao-Kai

Mechanical Engineering, AIES program

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Part 1: Im2col

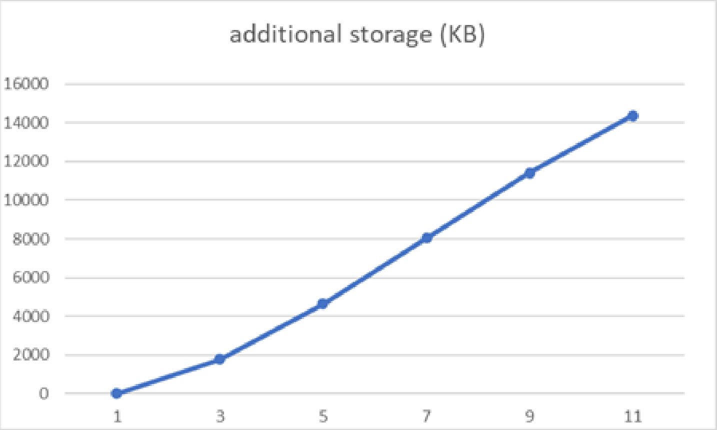


$$\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix} * \begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{bmatrix} = \begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}$$

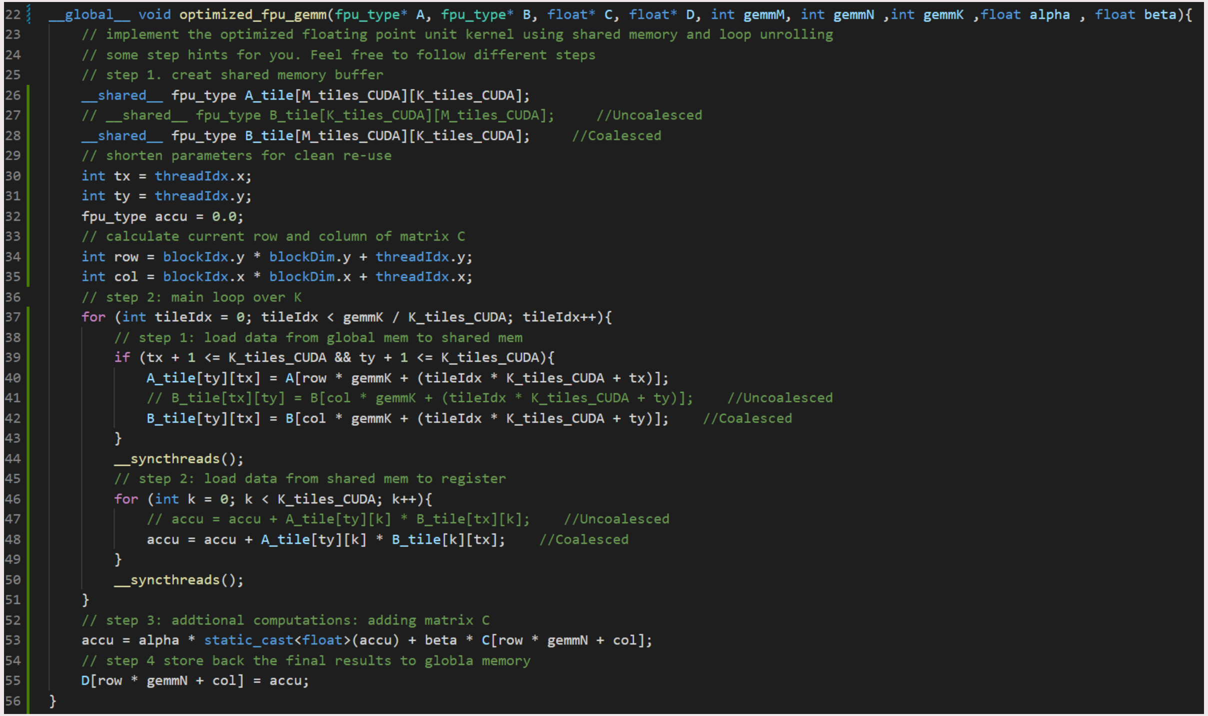
$$\begin{bmatrix}1 & 2 & 3 & 4\end{bmatrix} \times \begin{bmatrix}1 & 2 & 4 & 5\\ 2 & 3 & 5 & 6\\ 4 & 5 & 7 & 8\\ 5 & 6 & 8 & 9\end{bmatrix} = \begin{bmatrix}1 & 2 & 3 & 4\end{bmatrix}$$

$$original\ memory = H * W * M * N * \frac{4}{1024}\,(KB)$$

$$im2col\ memory = R * S * (H - R + 1) * (W - S + 1) * M * N * \frac{4}{1024}\,(KB)$$



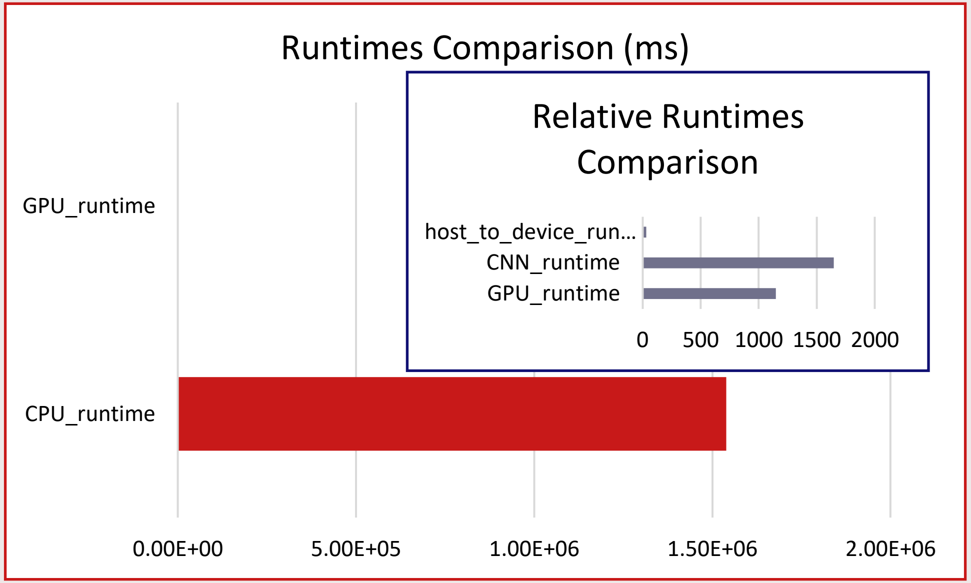
Part 2: GPU

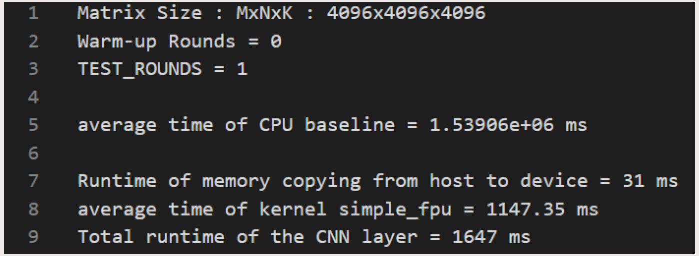


```
__global__ void optimized_fpu_gemm(fpu_type* A, fpu_type* B, float* C, float* D, int gemmM, int gemmN ,int gemmK ,float alpha , float beta){
    // implement the optimized floating point unit kernel using shared memory and loop unrolling
    // some step hints for you. Feel free to follow different steps
    // step 1. creat shared memory buffer
    __shared__ fpu_type A_tile[M_tiles_CUDA][K_tiles_CUDA];
    // __shared__ fpu_type B_tile[K_tiles_CUDA][M_tiles_CUDA];     //Uncoalesced
    __shared__ fpu_type B_tile[M_tiles_CUDA][K_tiles_CUDA];     //Coalesced
    // shorten parameters for clean re-use
    int tx = threadIdx.x;
    int ty = threadIdx.y;
    fpu_type accu = 0.0;
    // calculate current row and column of matrix C
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;
    // step 2: main loop over K
    for (int tileIdx = 0; tileIdx < gemmK / K_tiles_CUDA; tileIdx++){
        // step 1: load data from global mem to shared mem
        if (tx + 1 <= K_tiles_CUDA && ty + 1 <= K_tiles_CUDA){
            A_tile[ty][tx] = A[row * gemmK + (tileIdx * K_tiles_CUDA + tx)];
            // B_tile[tx][ty] = B[col * gemmK + (tileIdx * K_tiles_CUDA + ty)];    //Uncoalesced
            B_tile[ty][tx] = B[col * gemmK + (tileIdx * K_tiles_CUDA + ty)];    //Coalesced
        }
        __syncthreads();
        // step 2: load data from shared mem to register
        for (int k = 0; k < K_tiles_CUDA; k++){
            // accu = accu + A_tile[ty][k] * B_tile[tx][k];    //Uncoalesced
            accu = accu + A_tile[ty][k] * B_tile[k][tx];    //Coalesced
        }
        __syncthreads();
    }
    // step 3: addtional computations: adding matrix C
    accu = alpha * static_cast<float>(accu) + beta * C[row * gemmN + col];
    // step 4 store back the final results to globla memory
    D[row * gemmN + col] = accu;
}
```

Baseline CUDA core

CPU_runtime	GPU_runtime	CNN_runtime	host_to_device_runtime
1.54E+06	1147.35	1647	31





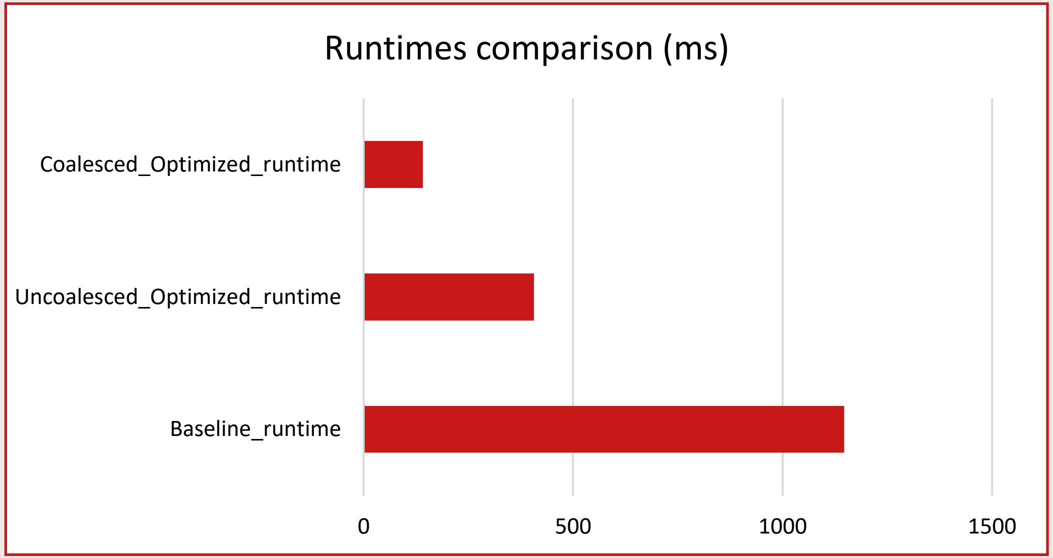
```
Matrix Size : MxNxK : 4096x4096x4096
Warm-up Rounds = 0
TEST_ROUNDS = 1

average time of CPU baseline = 1.53906e+06 ms

Runtime of memory copying from host to device = 31 ms
average time of kernel simple_fpu = 1147.35 ms
Total runtime of the CNN layer = 1647 ms
```

$$1540000 \div 1147.35 \approx 1342\ times\ faster$$

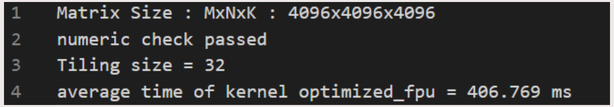
$$1540000 \div 1647 \approx 935\ times\ faster$$

Optimized CUDA core

Baseline_runtime	Uncoalesced_Optimized_runtime	Coalesced_Optimized_runtime
1147.35	406.769	141.204

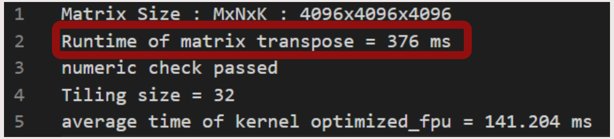


Uncoalesced tiling



```
Matrix Size : MxNxK : 4096x4096x4096
numeric check passed
Tiling size = 32
average time of kernel optimized_fpu = 406.769 ms
```

Coalesced tiling

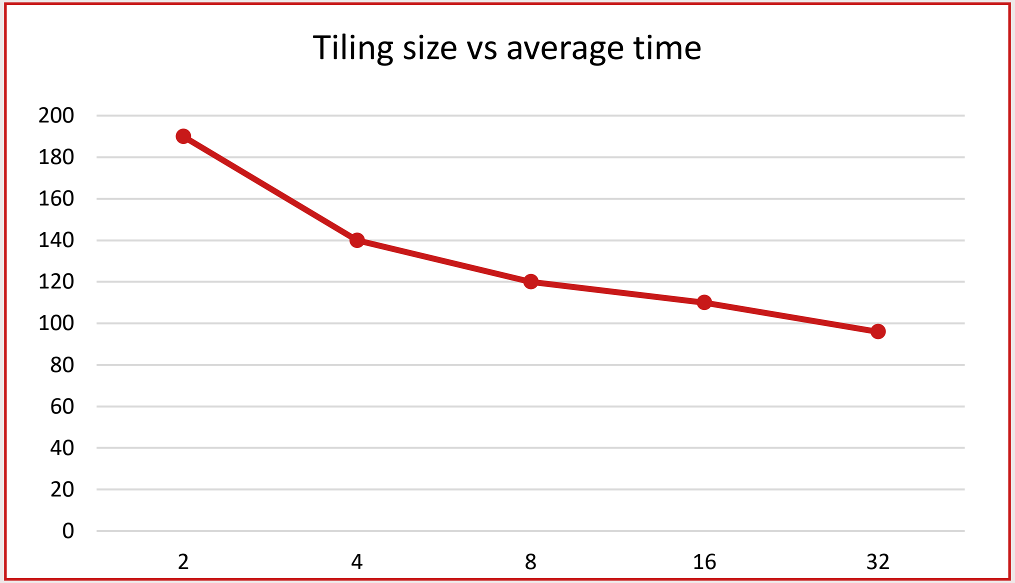


```
Matrix Size : MxNxK : 4096x4096x4096
Runtime of matrix transpose = 376 ms
numeric check passed
Tiling size = 32
average time of kernel optimized_fpu = 141.204 ms
```

$$1147.35 \div 406.769 \approx 3\ times\ faster$$

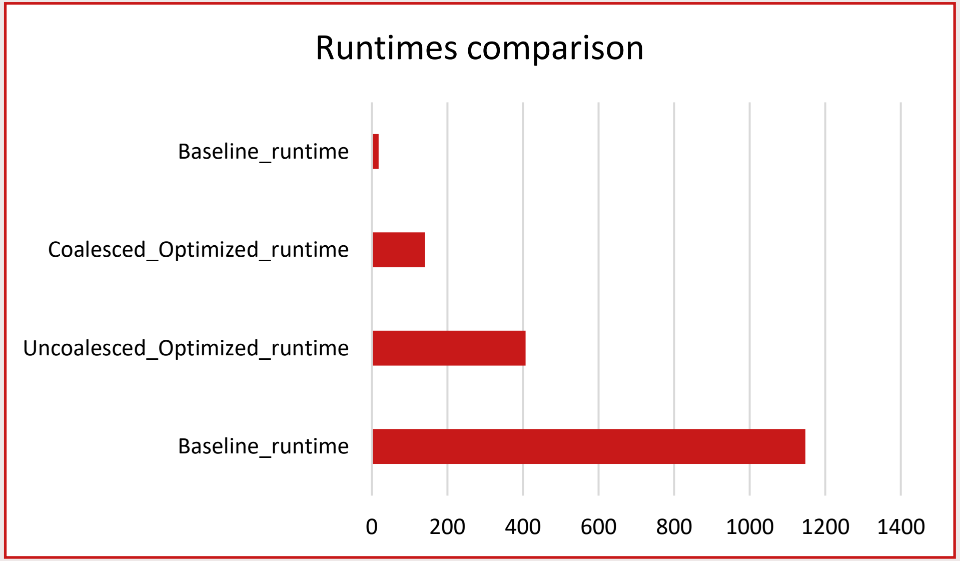
$$1147.35 \div 141.204 \approx 8\ times\ faster$$

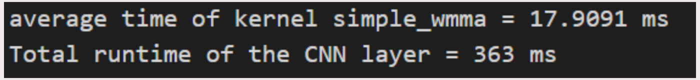
Tiling size vs Average time



Baseline Tensor core

CUDA			Tensor
Baseline_runtime	Uncoalesced_Optimized_runtime	Coalesced_Optimized_runtime	Baseline_runtime
1147.35	406.769	141.204	17.9





```
average time of kernel simple_wmma = 17.9091 ms
Total runtime of the CNN layer = 363 ms
```

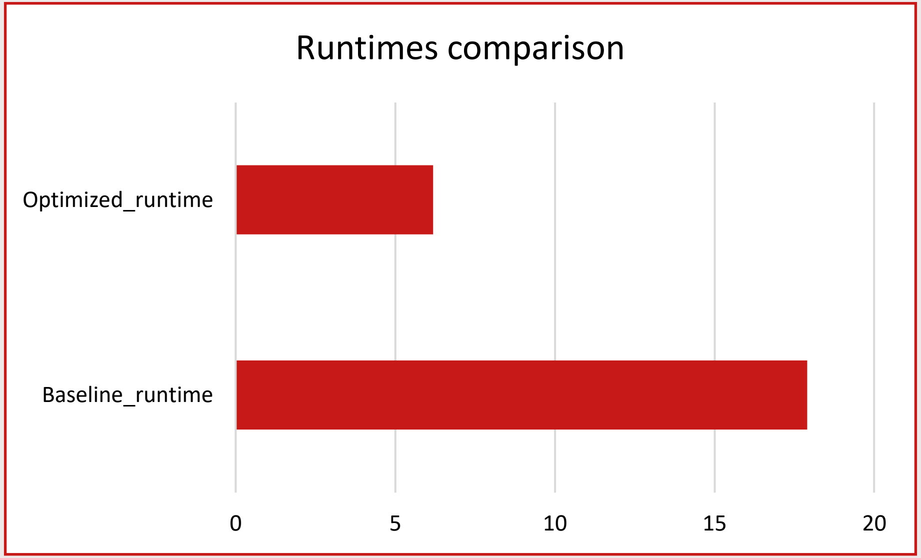
$$1147.35 \div 17.9 \approx 64\ times\ faster$$

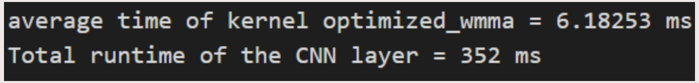
$$406.769 \div 17.9 \approx 23\ times\ faster$$

$$141.204 \div 17.9 \approx 8\ times\ faster$$

Optimized Tensor core (shared memory only)

Tensor	
Baseline_runtime	Optimized_runtime
17.9	6.18



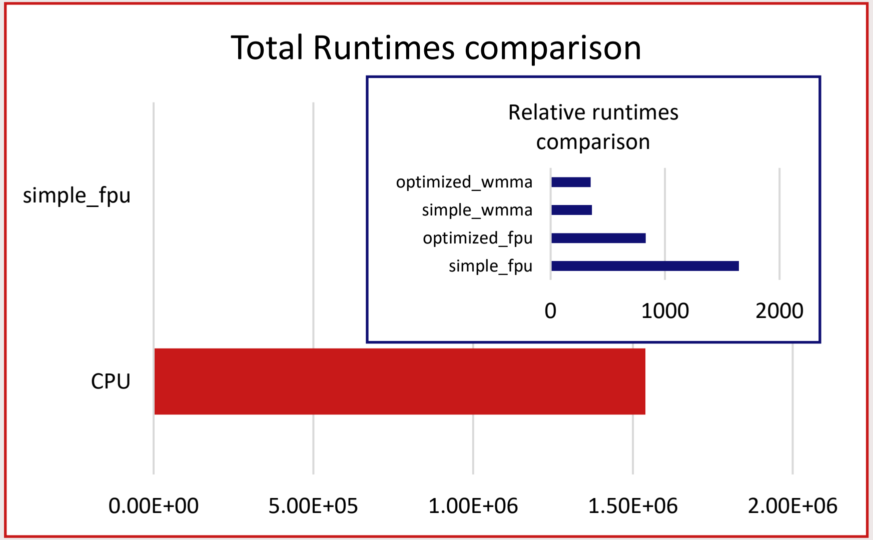


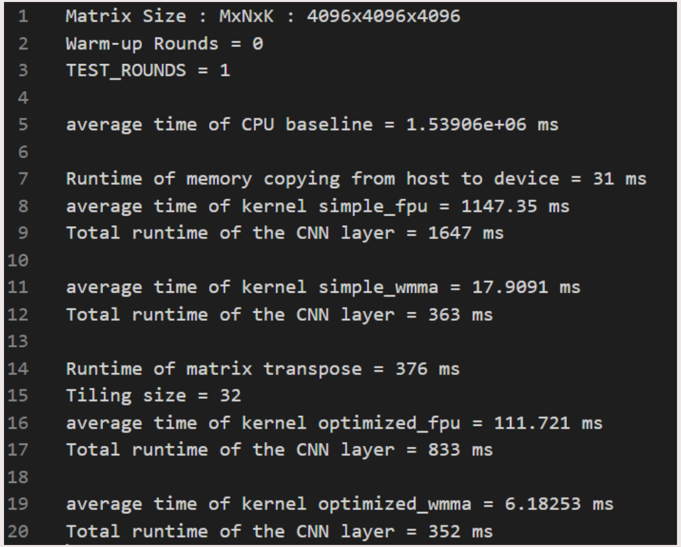
```
average time of kernel optimized_wmma = 6.18253 ms
Total runtime of the CNN layer = 352 ms
```

$$17.9 \div 6.18 \approx 3\ times\ faster$$

Comparison of total run time

Total Runtime				
CPU	simple_fpu	optimized_fpu	simple_wmma	optimized_wmma
1.54E+06	1647	833	363	352
Kernel Runtime				
1.54E+06	1147.35	111.72	17.9	6.18





```
Matrix Size : MxNxK : 4096x4096x4096
Warm-up Rounds = 0
TEST_ROUNDS = 1

average time of CPU baseline = 1.53906e+06 ms

Runtime of memory copying from host to device = 31 ms
average time of kernel simple_fpu = 1147.35 ms
Total runtime of the CNN layer = 1647 ms

average time of kernel simple_wmma = 17.9091 ms
Total runtime of the CNN layer = 363 ms

Runtime of matrix transpose = 376 ms
Tiling size = 32
average time of kernel optimized_fpu = 111.721 ms
Total runtime of the CNN layer = 833 ms

average time of kernel optimized_wmma = 6.18253 ms
Total runtime of the CNN layer = 352 ms
```