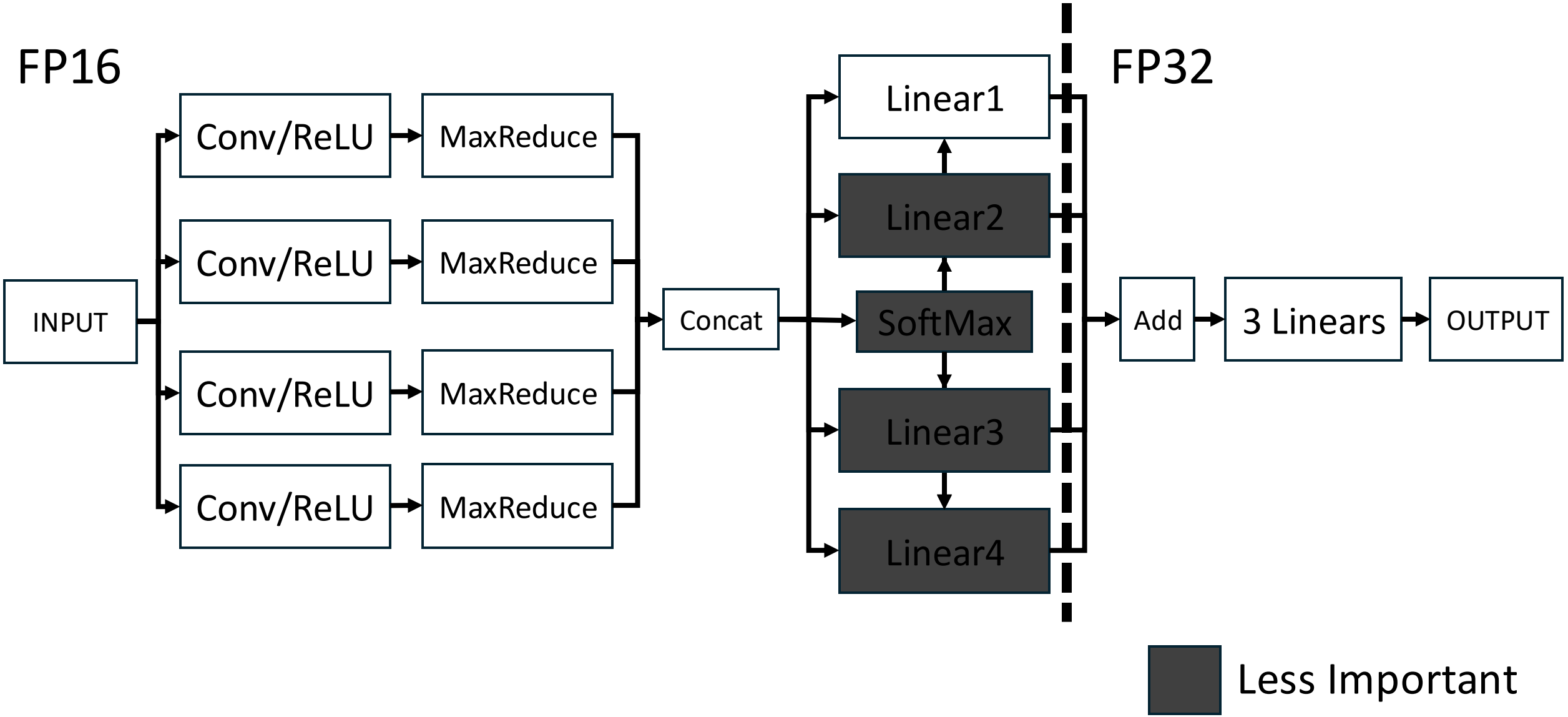
MoE Model Optimization

HoJoon Kim

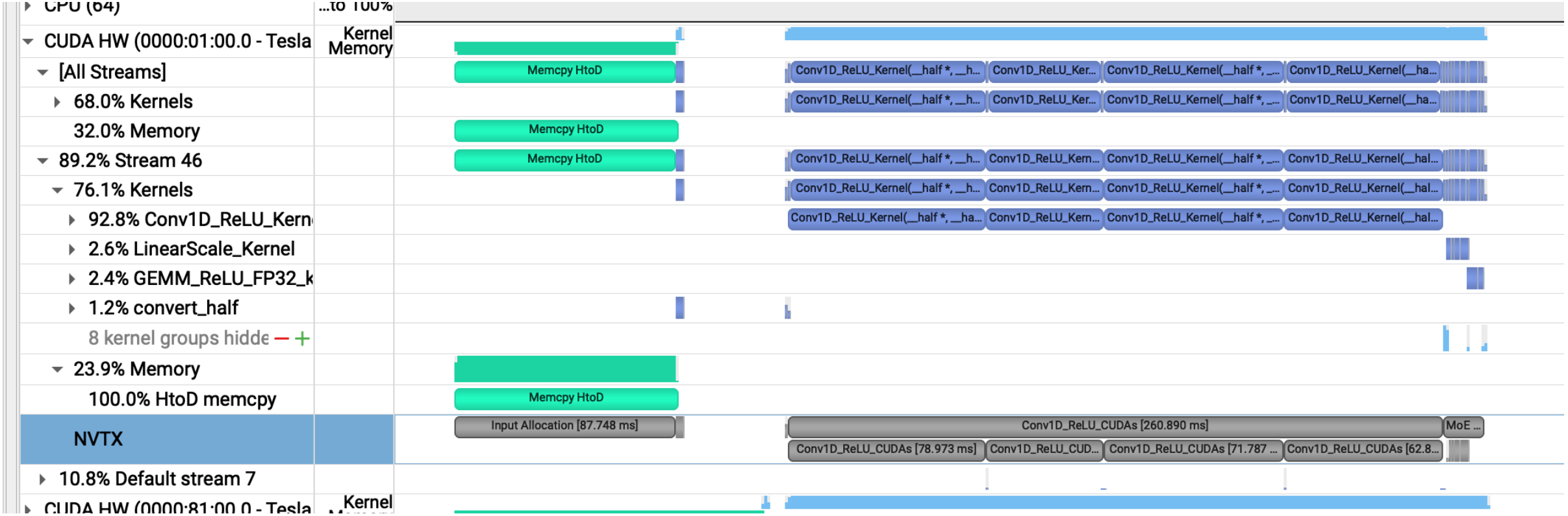
BATCH=16384 / GPU=4

THROUGHPUT = 40000 sentences/sec

Model Architecture



Model Architecture



Pinned Memory

```
int N = BATCH;
for(int g = 0; g < NGPU; g++) {
  CHECK_CUDA(cudaSetDevice(g));
  CHECK_CUDA(cudaMemcpyAsync(input_gpu[g], inputs + g * N * SEQ_LEN * 4096, N *
}
```

Input

```
GEMM_FP32_kernel<<<gridDim_3,blockDim_3,0,input_streams[g]>>>( linear2_w_gpu[g], linear1
        linear2_a_gpu[g], 2, BATCH, 512);

nvtxRangePop();
convert_transpose_float<<<BATCH/128,256,0,input_streams[g]>>>(linear2_a_gpu[g], outputs
}
```

Output

Easy memcopy!

FP16/FP32 Version kernels

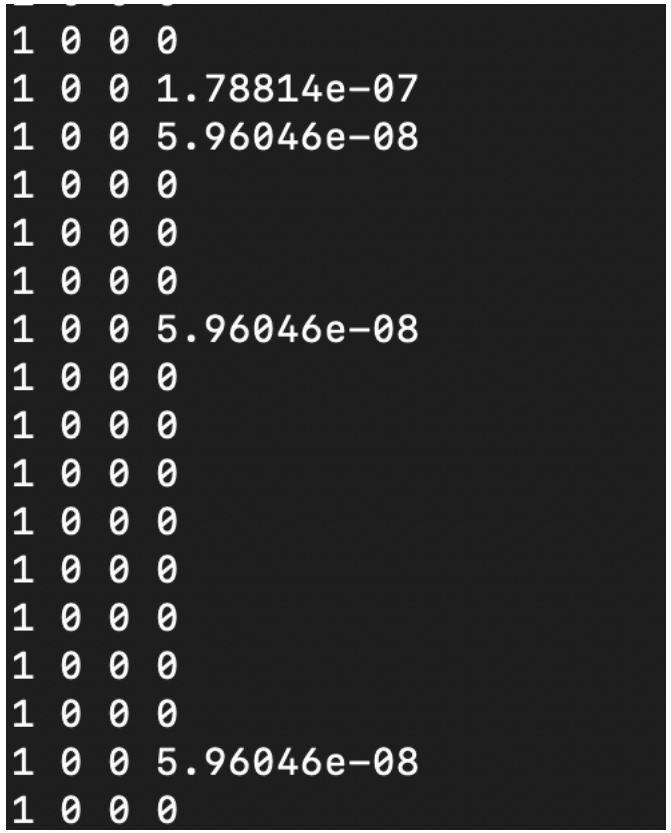
```
__global__ void GEMM_Scale_kernel(half* A, h
  int j = blockIdx.x*blockDim.x + threadIdx.
```

Half Precision Kernel

```
__global__ void GEMM_Scale_FP32_kernel(float* A, f
  int j = blockIdx.x*blockDim.x + threadIdx.x;
```

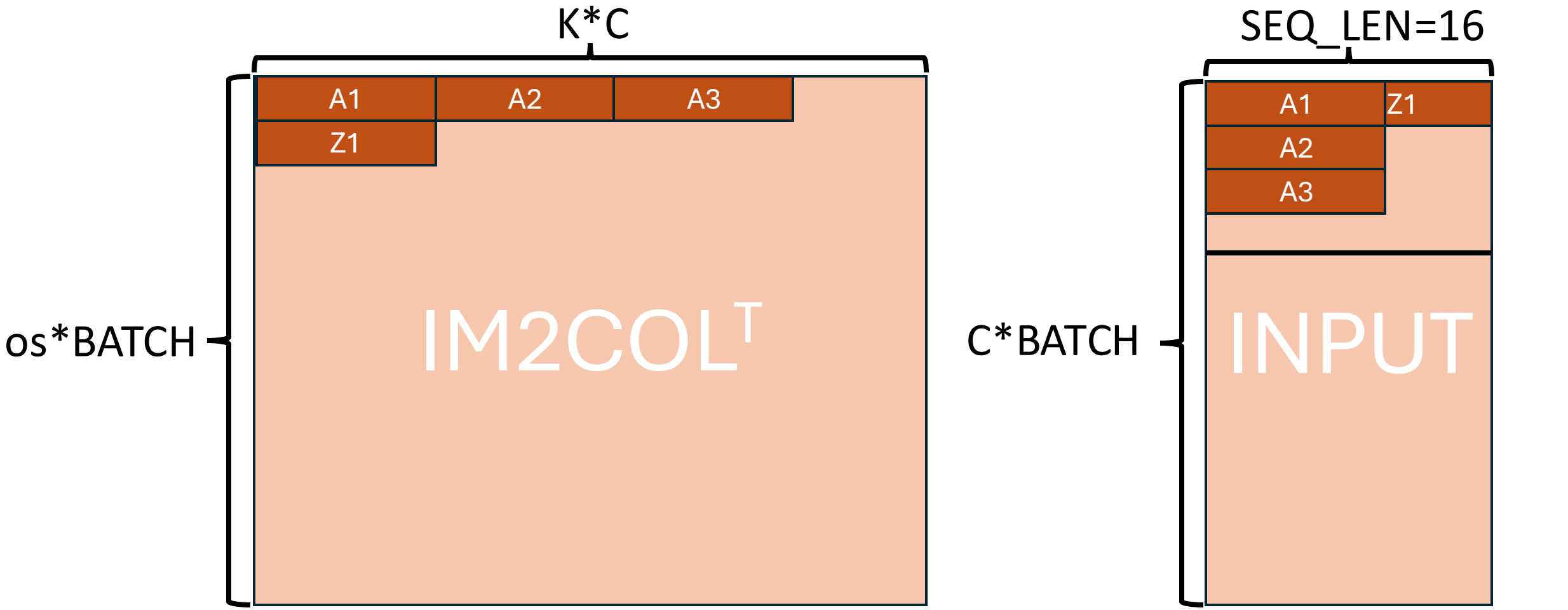
Full Precision Kernel

Not all layers are Important



Softmax Output

Implicit IM2COL



$$IM2COL^T[y,x] = INPUT[y/os * C + x/K, y\%os + x\%K]$$

```
int in_t_x = g_k + fp16_id*2;
CONVERT_HALF2(sharedA[c * blockDim.x * 2 + t_i*2]) = CONVERT_HALF2(w[(g_i + 8*c + fp16_line)*K*C + in_t_x]);
}
```

Conv Weight Load

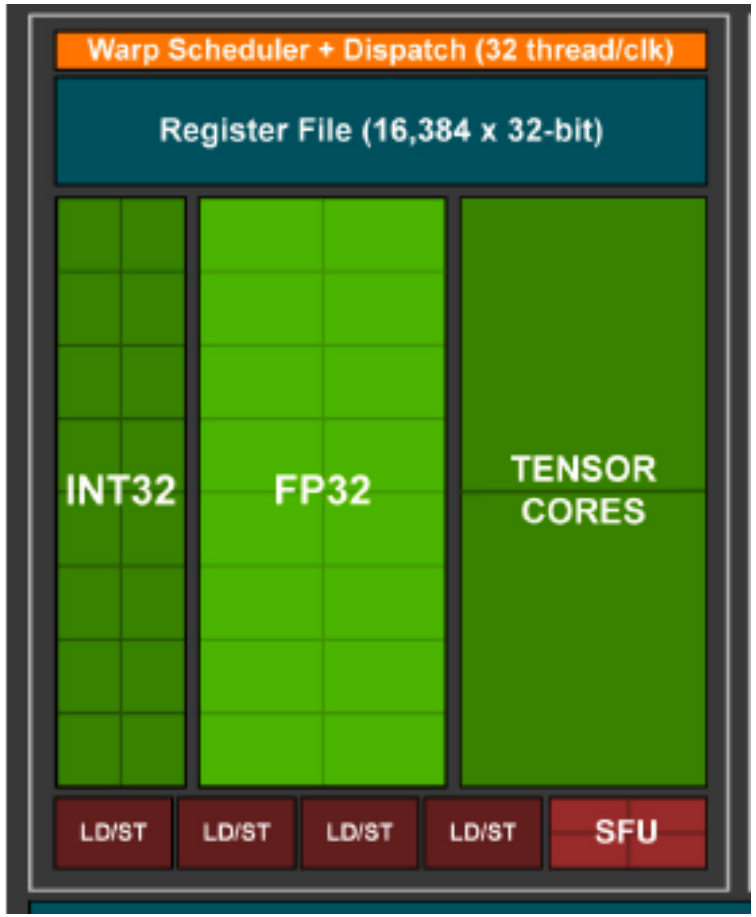
```
for(int c = 0; c < load_count*2; c+=1){
  int in_t_x = g_k + laneId;
  int in_t_y = g_j + warpId + 4*c;
  int temp1 = in_t_x/K;
  int temp4 = in_t_x%K;
  int temp2 = in_t_y/os;
  int temp3 = in_t_y%os;
  int in_y = temp2 * C + temp1;
  int in_x = temp3 + temp4;

  sharedB[c * blockDim.x + t_i] = in[in_y*s + in_x];
}
__syncthreads();
```

Direct Load from Input Sequencs

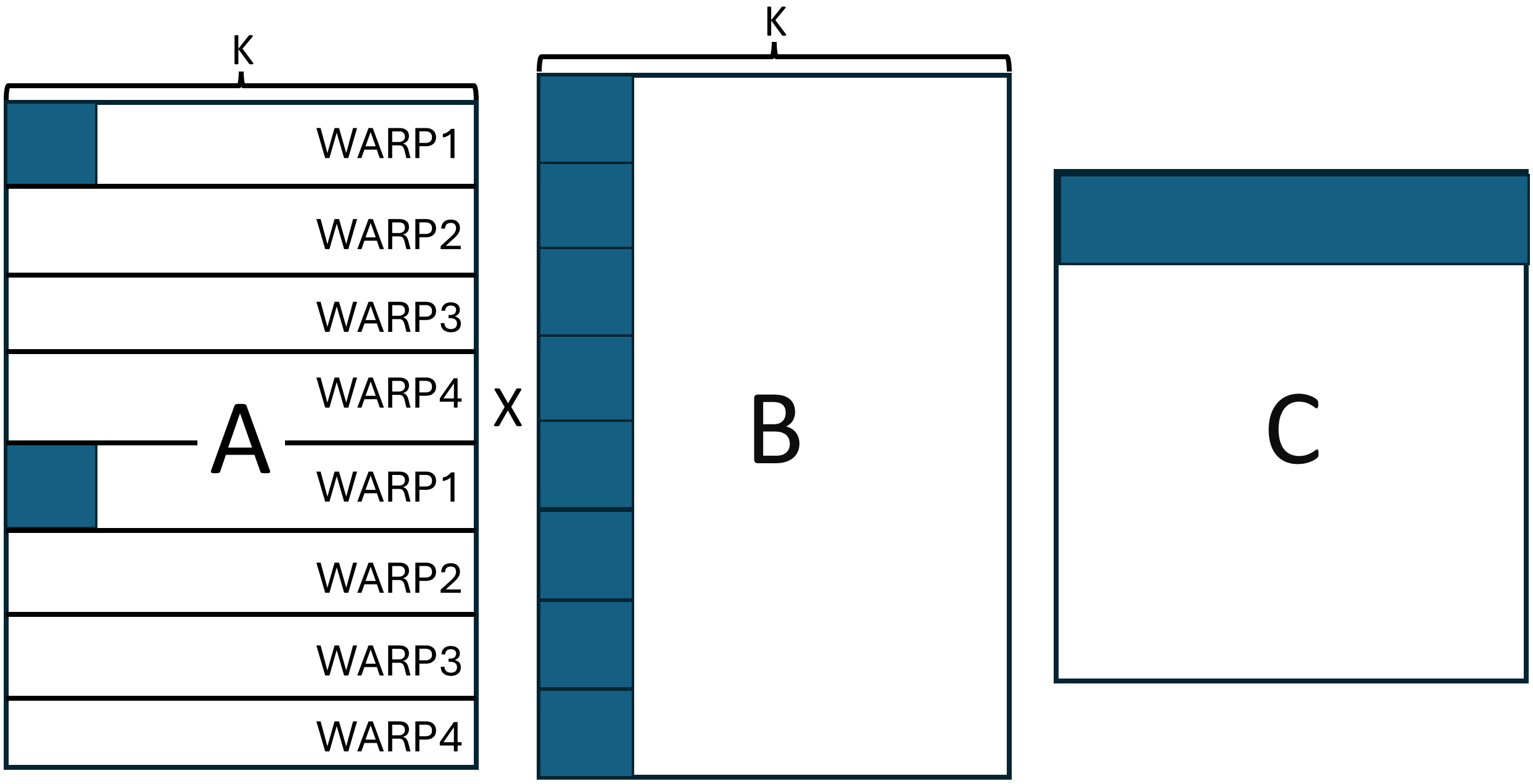
But is it actually good?

GEMM




```
    }
    __syncthreads();
    for (int colA = 0; colA < BLK_SZ_K; colA += WMMA_M) {
      wmma::load_matrix_sync(a_frag[0], sharedA + colA + mem_A, BLK_SZ_K);
      wmma::load_matrix_sync(a_frag[1], sharedA + colA + temp5 + mem_A, BLK_SZ_K);
      for(int rowB = 0; rowB < 8; rowB += 1) {
        wmma::load_matrix_sync(b_frag[rowB], sharedB + colA + rowB*BLK_SZ_K*WMMA_M, BLK_SZ_K);
        wmma::mma_sync(c_frag[rowB], a_frag[0], b_frag[rowB], c_frag[rowB]);
        wmma::mma_sync(c_frag[rowB+8], a_frag[1], b_frag[rowB], c_frag[rowB+8]);
      }
      __syncthreads();
    }
}
```

Utilize both Tensor Cores


K
WARP1
WARP2
WARP3
WARP4
A
WARP4
WARP1
WARP2
WARP3
WARP4
X
K
B
C