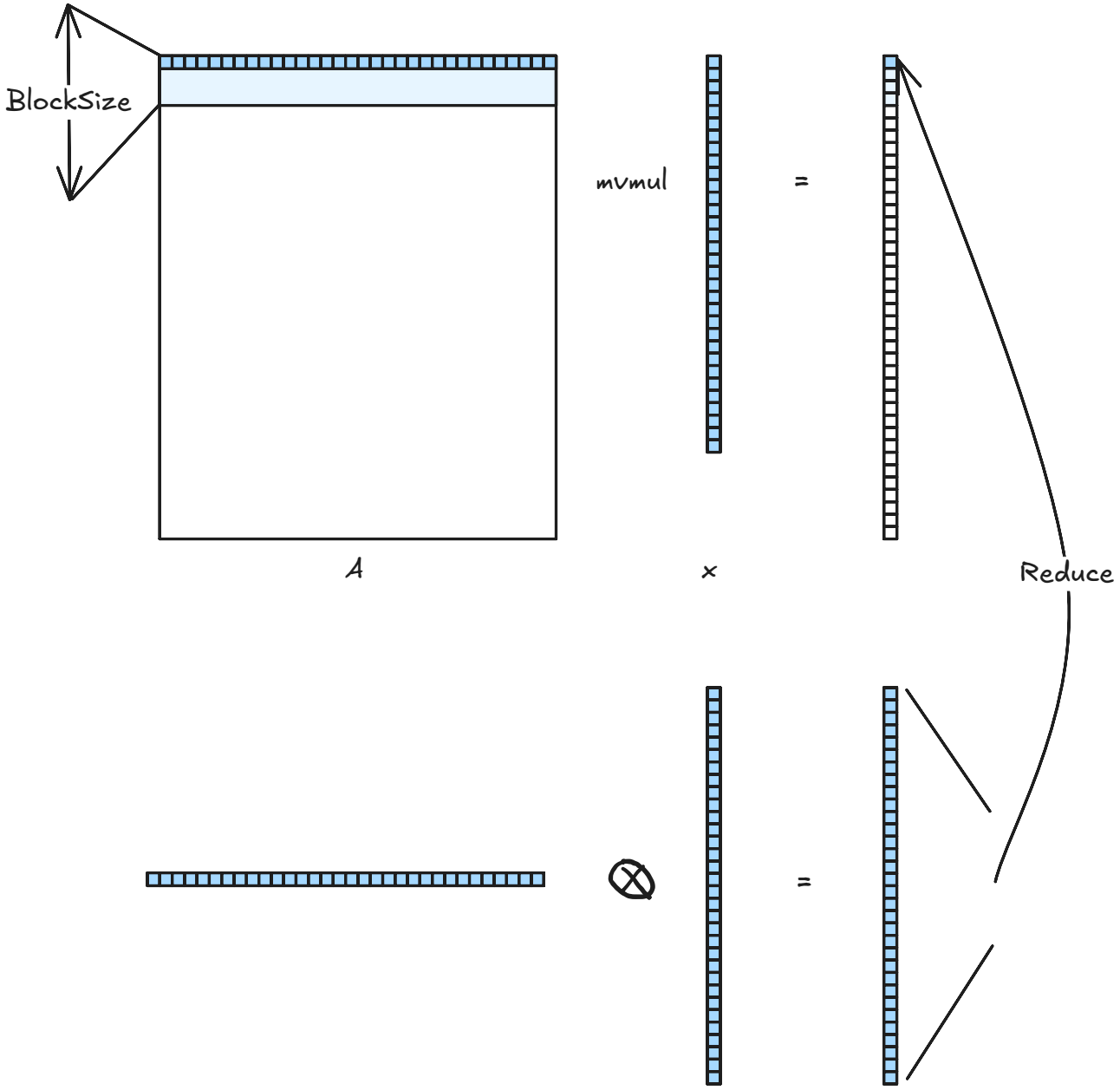
SGEMV

SGEMV的优化手段比较有限，原则上保持thread的高利用率即可。让warp中的每一个thread都有workload即可。这个可以基于矩阵的列数粗略分成3个情况。但是对所有情况最后的计算工作都是对应元素相乘再warpReduce。这里给出warpReduce代码：

```
template <unsigned int WARPSIZE>
__device__ __forceinline__ float warpReduce(float sum)
{
    if (WARPSIZE >= 32)
        sum += __shfl_down_sync(0xffffffff, sum, 16);
    if (WARPSIZE >= 16)
        sum += __shfl_down_sync(0xffffffff, sum, 8);
    if (WARPSIZE >= 8)
        sum += __shfl_down_sync(0xffffffff, sum, 4);
    if (WARPSIZE >= 4)
        sum += __shfl_down_sync(0xffffffff, sum, 2);
    if (WARPSIZE >= 2)
        sum += __shfl_down_sync(0xffffffff, sum, 1);
    return sum;
}
```

sgemv_v0_32

这个版本适用矩阵列数$32 \leq N < 128$，此时每个线程负责一个矩阵中元素的对应乘。



对于大于32列宽的矩阵，我们可以让每一个warp在轴上横向移动。
代码：

```
__global__ void sgemv0(float *__restrict__ A, float *__restrict__
x, float *__restrict__ y, const int M, const int N)
{
    const int tid_x = threadIdx.x;
    const int tid_y = threadIdx.y;

    const int WARPSIZE = 32;
    int lane_id = tid_x % WARPSIZE;
    int cur_row = blockDim.y * blockIdx.x + tid_y;

    if (cur_row < M)
    {
```

```
        float sum = 0.0f;
#pragma unroll
        for (int i = 0; i < N; i += WARPSIZE)
        {
            int cur_col = i + lane_id;
            sum += A[cur_row * N + cur_col] * x[cur_col];
        }
        sum = warpReduce<WARPSIZE>(sum);
        if (lane_id == 0)
            y[cur_row] = sum;
    }
}
```

sgemv_v1_float4

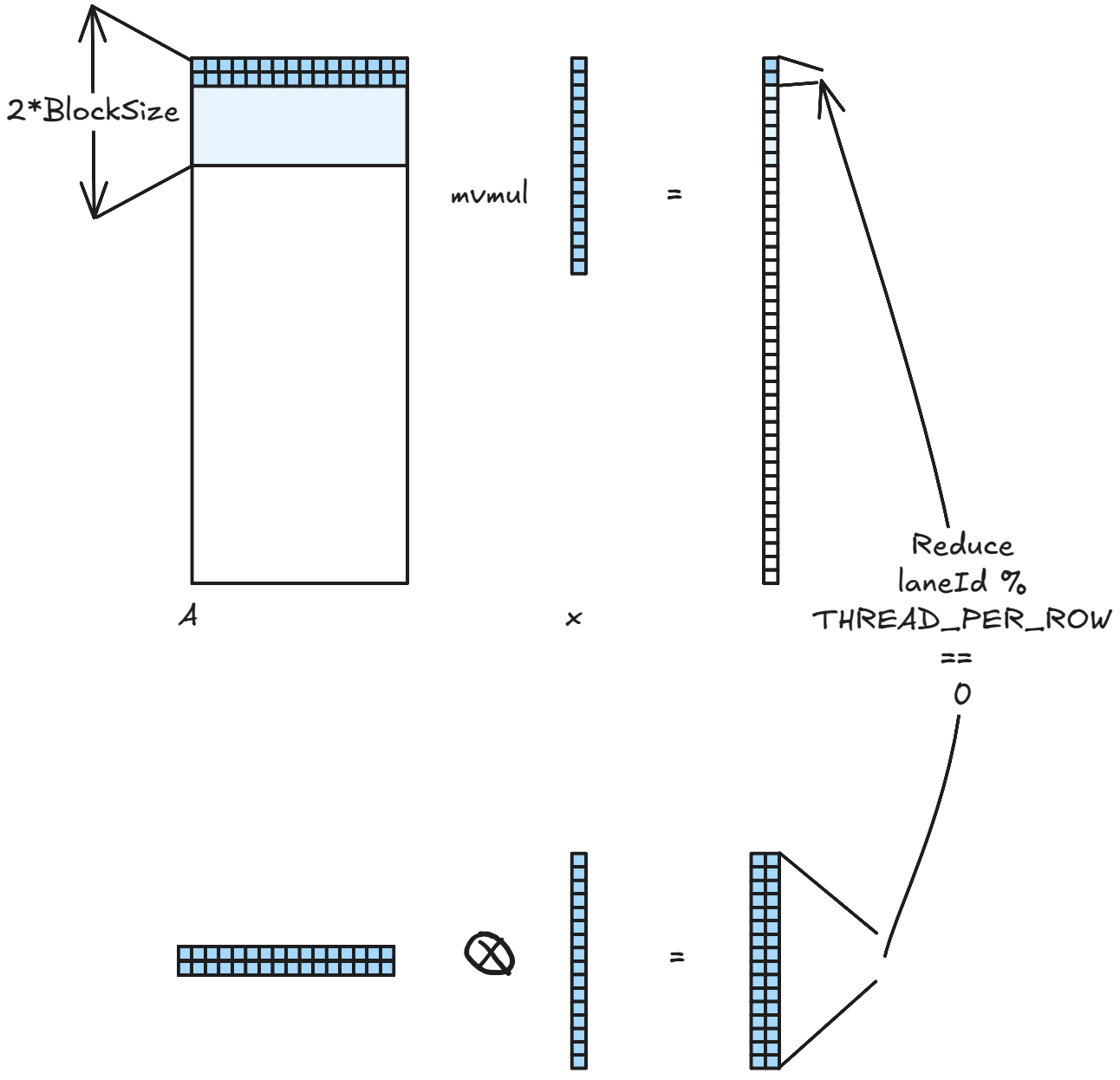
这个版本通过使用float4变量读写，增加访存效率。自然我们可以期望$N \leq 128$。代码：

```
__global__ void sgemv0(float *__restrict__ A, float *__restrict__
x, float *__restrict__ y, const int M, const int N)
{
    const int tid_x = threadIdx.x;
    const int tid_y = threadIdx.y;

    const int WARPSIZE = 32;
    int lane_id = tid_x % WARPSIZE;
    int cur_row = blockDim.y * blockIdx.x + tid_y;

    if (cur_row < M)
    {
        float sum = 0.0f;
#pragma unroll
        for (int i = 0; i < N; i += 4 * WARPSIZE)
        {
            int cur_col = i + lane_id * 4;
            float4 cur_col_vec = FETCH_FLOAT4(x[cur_col]);
            float4 cur_row_vec = FETCH_FLOAT4(A[cur_row * N +
cur_col]);
            sum += cur_col_vec.x * cur_row_vec.x;
            sum += cur_col_vec.y * cur_row_vec.y;
            sum += cur_col_vec.z * cur_row_vec.z;
            sum += cur_col_vec.w * cur_row_vec.w;
        }
```

```
        sum = warpReduce<WARPSIZE>(sum);
        if (lane_id == 0)
            y[cur_row] = sum;
    }
}
```

sgemv_v2_16

对于列宽较小的矩阵，我们需要让一个warp负责多行，这时需要单独计算索引。



代码：

```
__global__ void sgemv0(float *__restrict__ A, float *__restrict__
x, float *__restrict__ y, const int M, const int N)
{
    const int tid_x = threadIdx.x;
```

```
    const int tid_y = threadIdx.y;

    const int WARPSIZE = 32;
    int lane_id = tid_x % WARPSIZE;
    const int THREAD_PER_ROW = WARPSIZE / ROW_PER_WARP;
    int cur_row = (blockDim.y * blockIdx.x + tid_y) * ROW_PER_WARP
+ lane_id / THREAD_PER_ROW;

    if (cur_row < M)
    {
        float sum = 0.0f;
        int cur_col = lane_id % THREAD_PER_ROW;
        sum += A[cur_row * N + cur_col] * x[cur_col];
        sum = warpReduce<THREAD_PER_ROW>(sum);
        if (cur_col == 0)
            y[cur_row] = sum;
    }
}
```