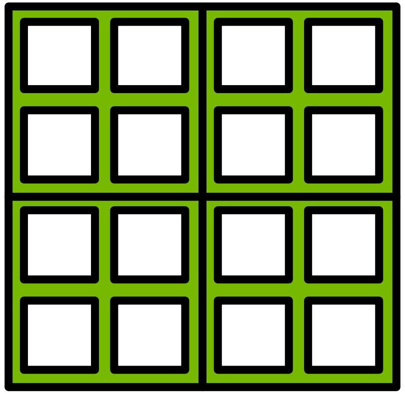
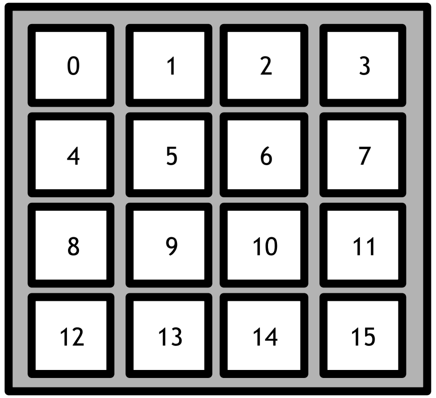
使用共享内存来支持合并内存访问

我们将研究矩阵转置，目的是演示如何**使用共享内存来**协助实现全局内存的双**向合并数据**传输

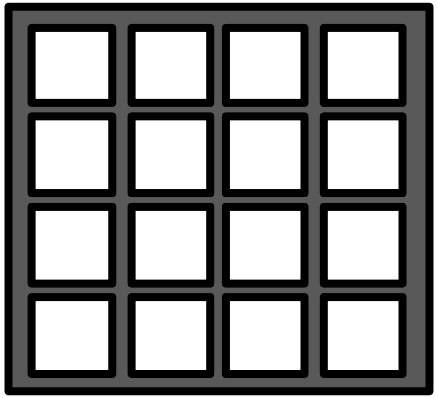
这里我们有一个 (2, 2) 的网格，其中每个块包含 (2, 2) 个线程。还有 (4,4) 的输入和输出矩阵。

网格

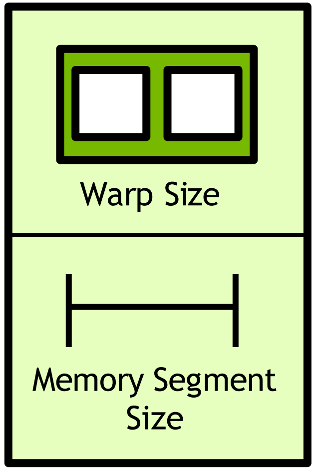




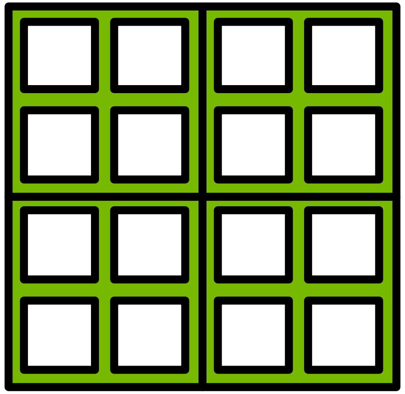
输入



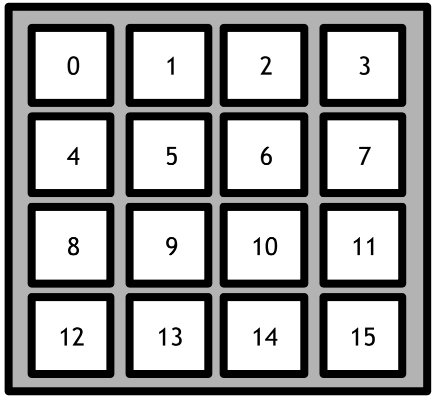
输出



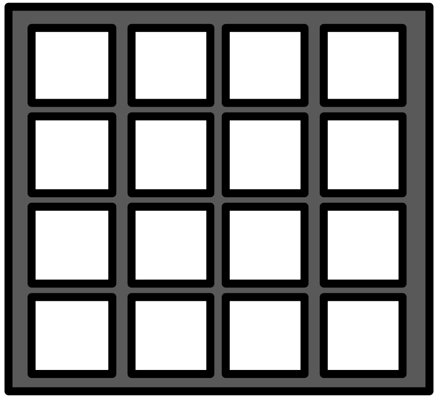
网格



在这个演示中，我们将一个 Warp 定义为 2 个线程，一个内存段定义为 2 个数据元素宽。

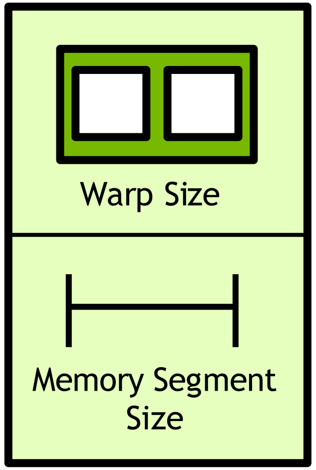


输入

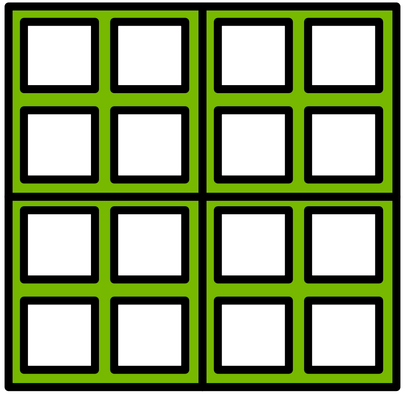


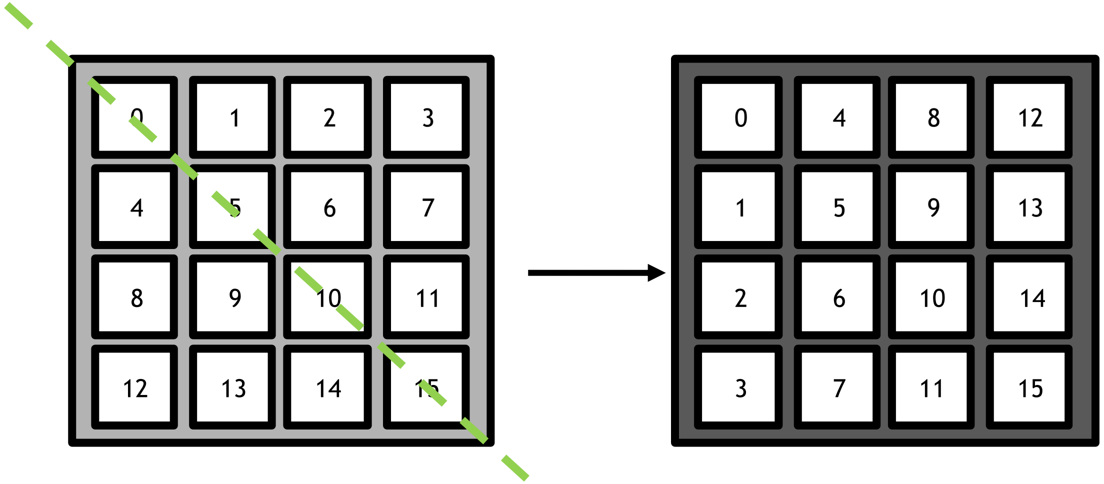
输出

我们的目标是通过围绕对角线旋转所有元素来对输入进行转置，并将转置的元素写入到输出矩阵。

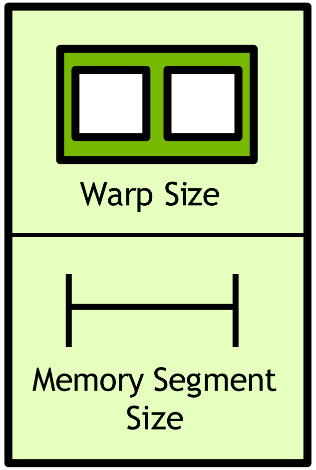


网格

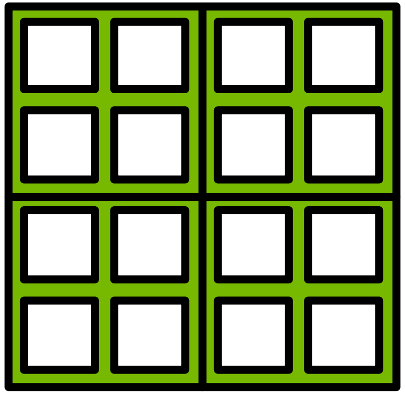




输入　　输出

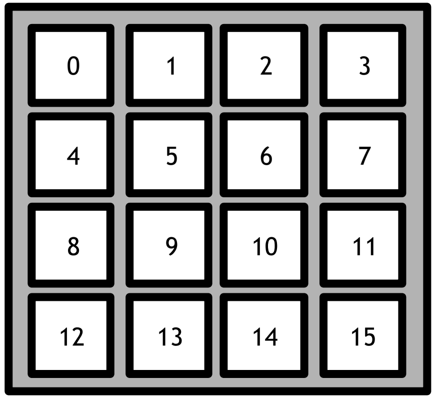


网格

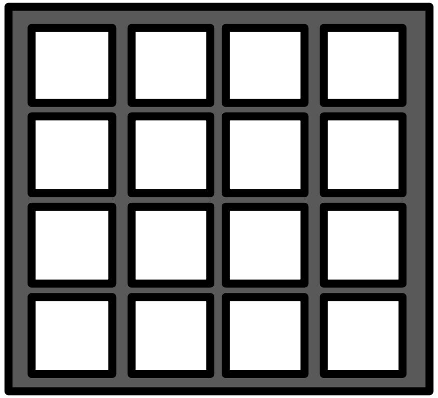


一种简单的方法是启动一个网格，其内的线程数等于输入元素的个数。让每个线程读取 1 个元素，然后将其写入转置位置的输出。

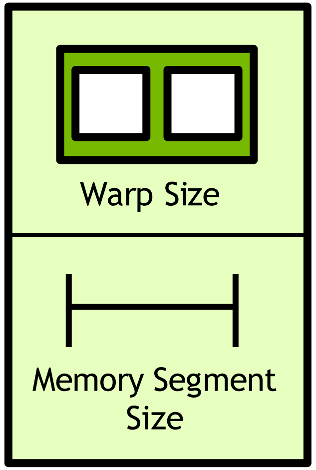
```
x, y = cuda.grid(2)

out[x][y] = in[y][x]
```



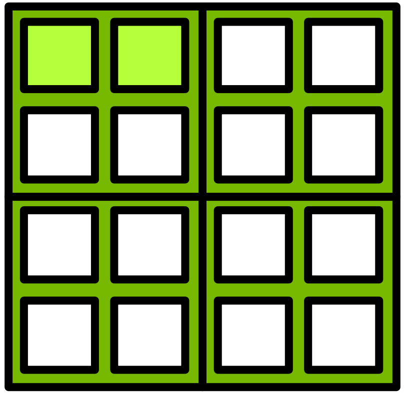
输入



输出

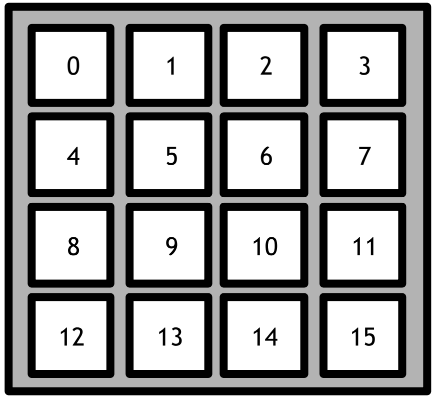


网格

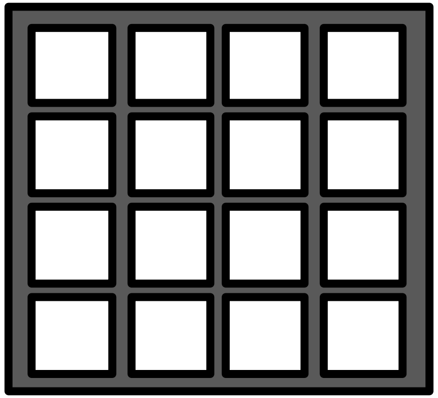


观察单个Warp的行为，内存读取是否是**合并的？**让我们深入回答这个问题。

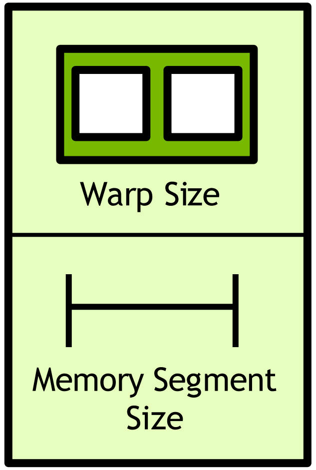
```
x, y = cuda.grid(2)

out[x][y] = in[y][x]
```



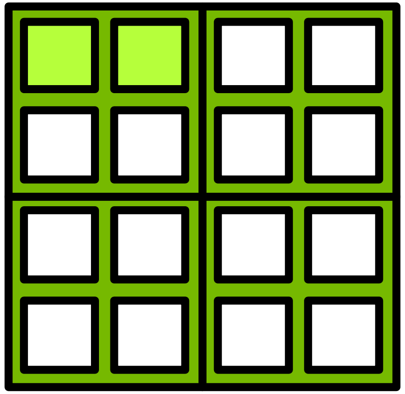
输入



输出

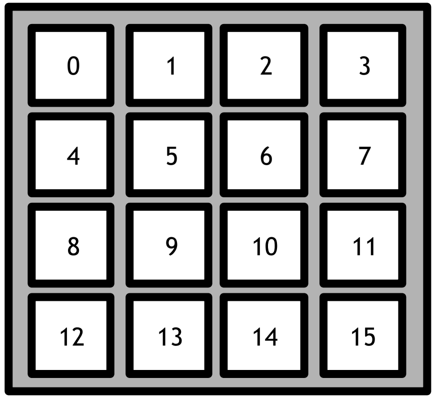


网格

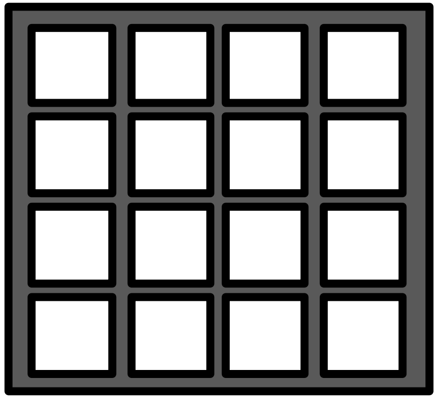


重写线程索引变量的计算公式， 我们清楚地看到， 同一 warp 中的连续线程在 x 轴方向上是相邻的。

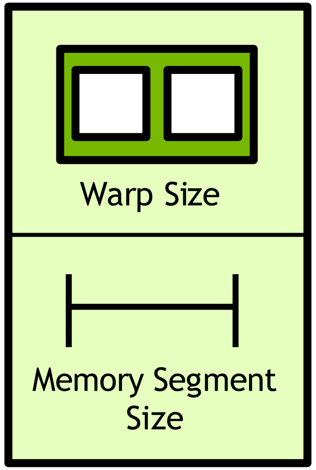
```
x = blockIdx.x * blockDim.x + threadIdx.x

y = blockIdx.y * blockDim.y + threadIdx.y

out[x][y] = in[y][x]
```



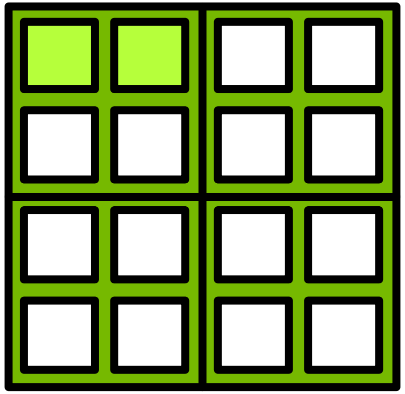
输入



输出

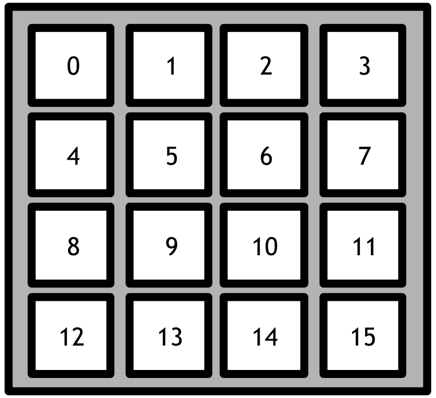


网格

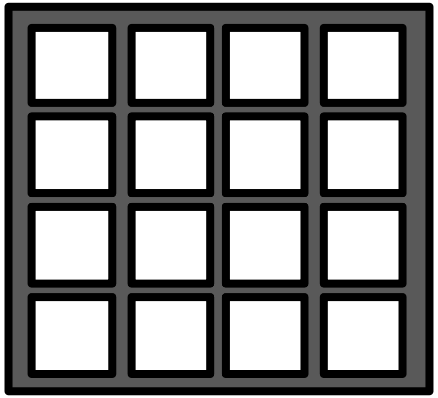


此外，这些连续线程将从数据元素连续的输入行中读取元素

```
x = blockIdx.x * blockDim.x        +
threadIdx.x

y = blockIdx.y * blockDim.y        +
threadIdx.y

out[x][y] = in[y][x]
```

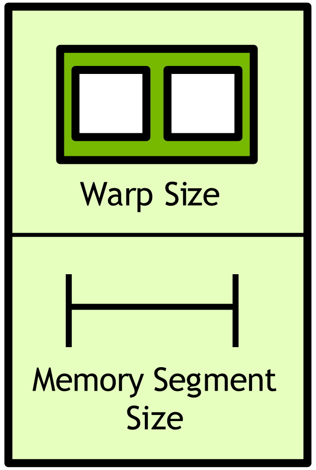


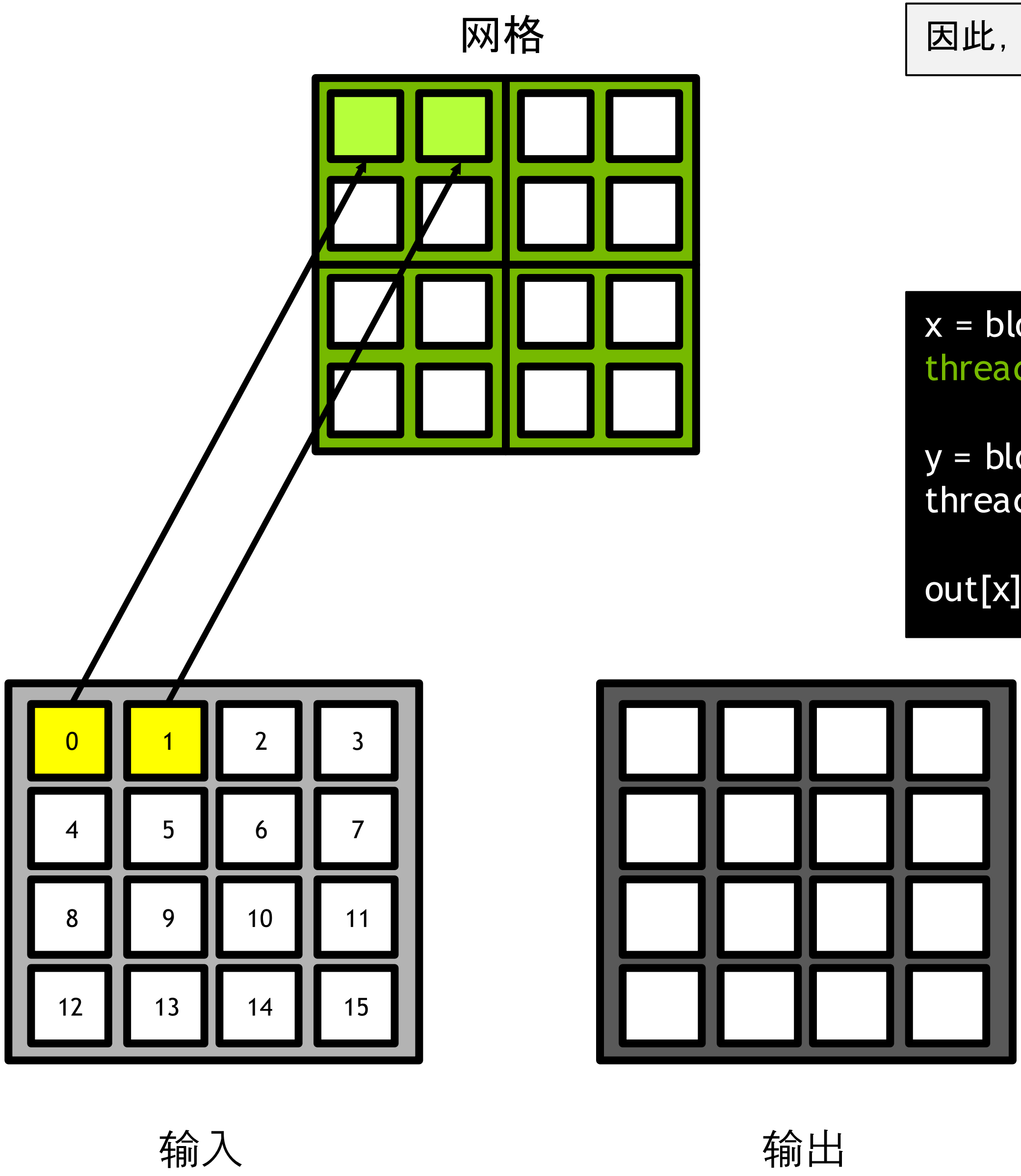
输入



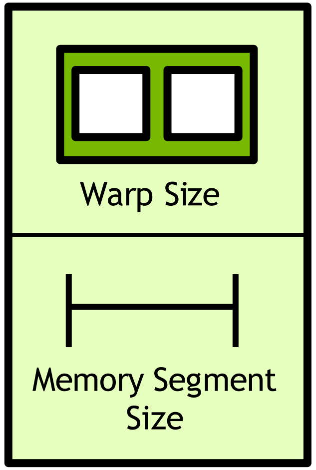
输出

因此，对输入数据的读取是合并的。

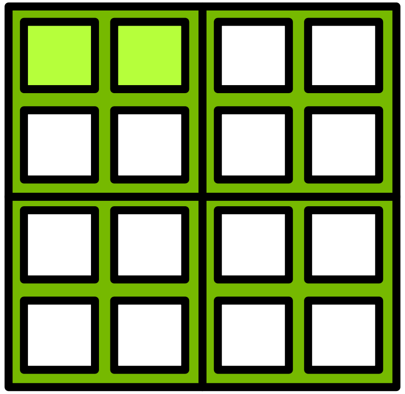





```
x = blockIdx.x * blockDim.x + threadIdx.x

y = blockIdx.y * blockDim.y + threadIdx.y

out[x][y] = in[y][x]
```

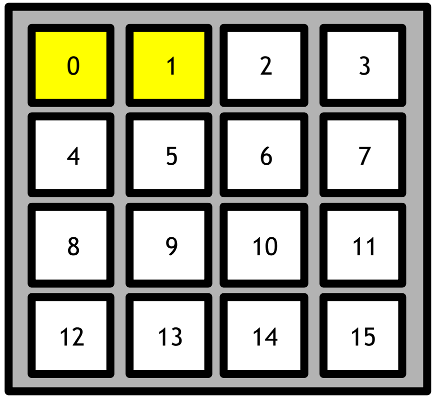


网格

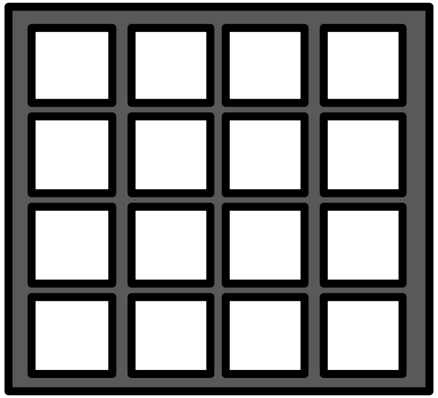


那么，这个 Warp 里的线程在把数据写入输出矩阵时是合并的吗？

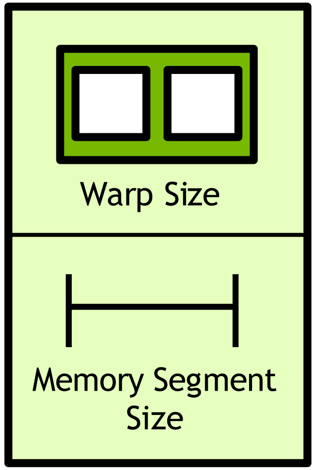
```
x = blockIdx.x * blockDim.x         +
threadIdx.x

y = blockIdx.y * blockDim.y         +
threadIdx.y

out[x][y] = in[y][x]
```



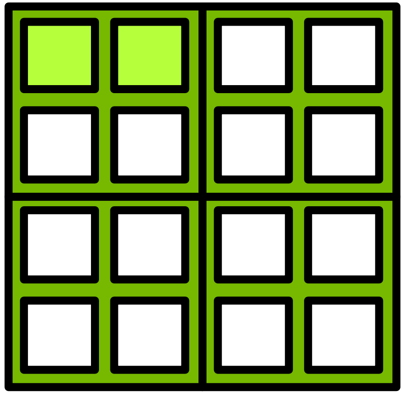
输入



输出

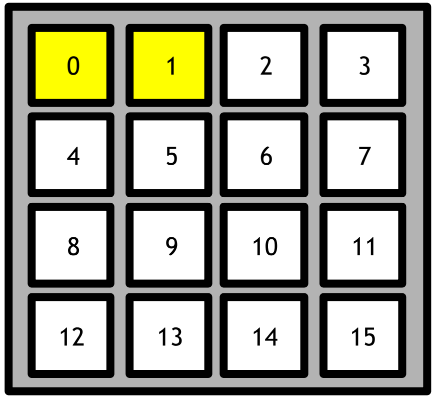


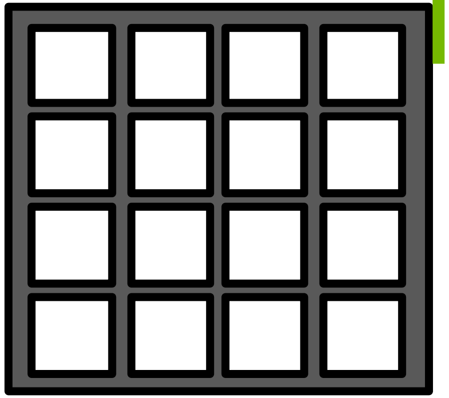
网格



我们看到同一 Warp 中的连续线程将沿着输出矩阵中的列写入数据。

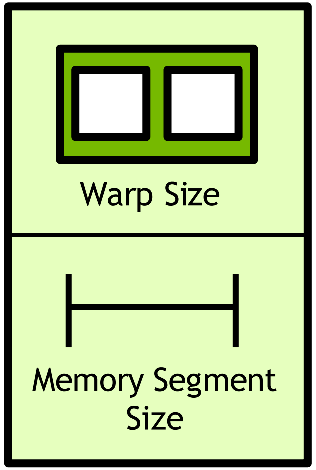
```
x = blockIdx.x * blockDim.x + threadIdx.x

y = blockIdx.y * blockDim.y + threadIdx.y

out[x][y] = in[y][x]
```

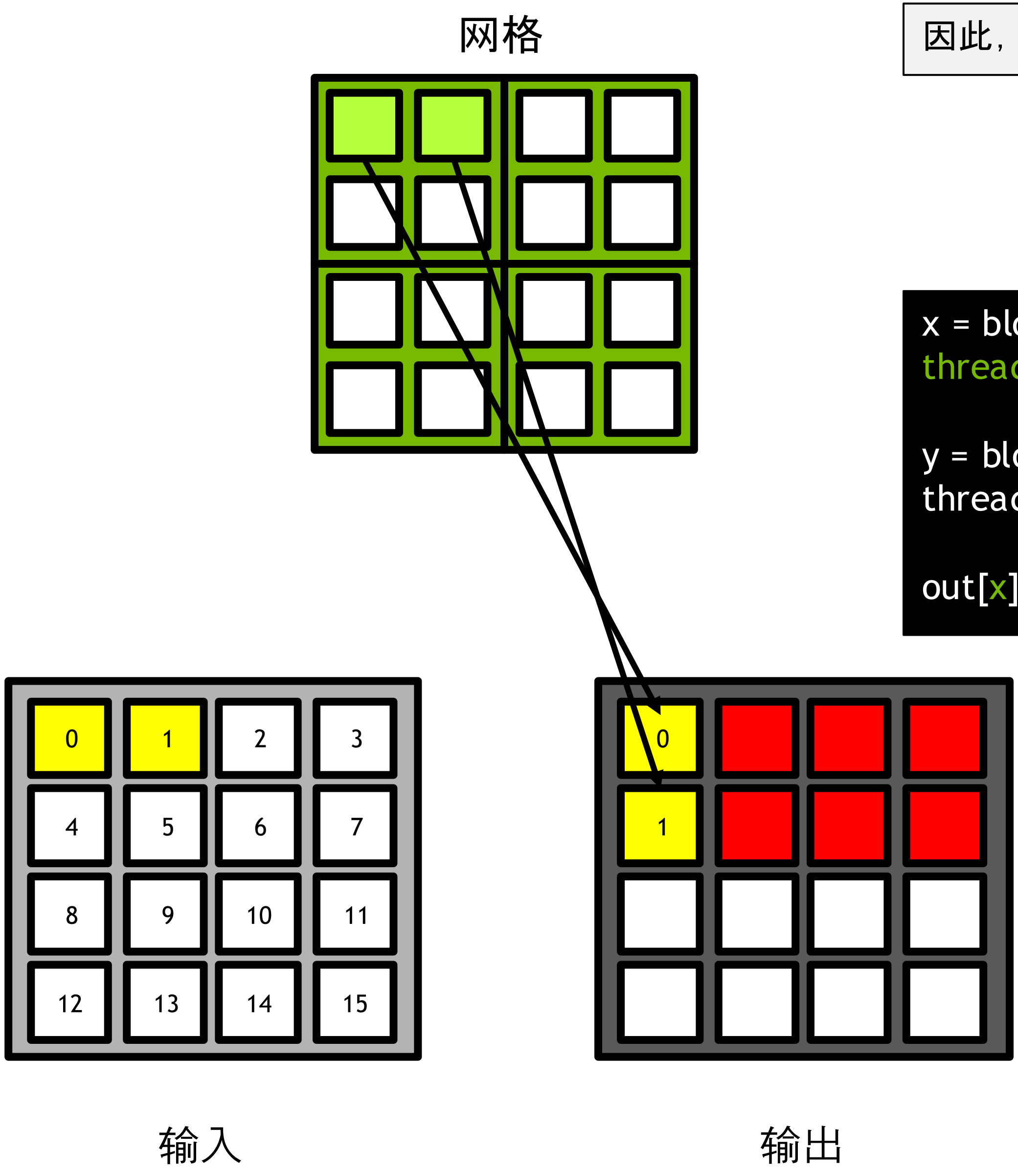




输入

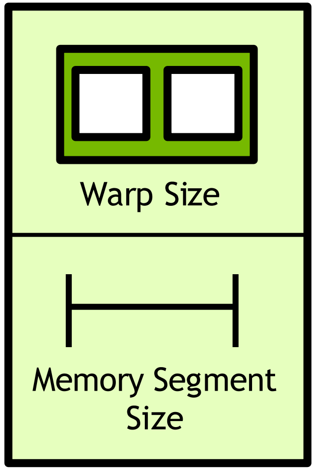
输出

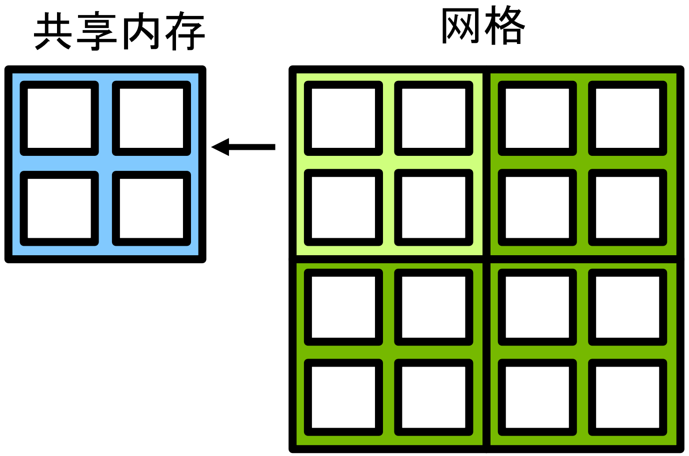




因此，写入数据时不是合并写。

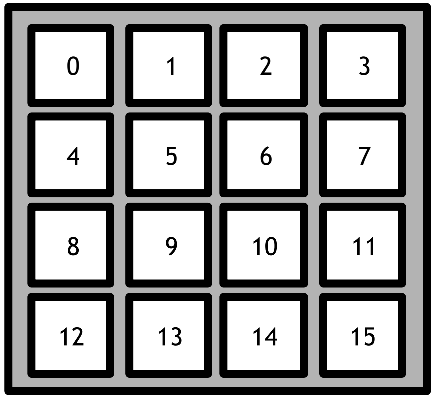
```
x = blockIdx.x * blockDim.x        +
threadIdx.x

y = blockIdx.y * blockDim.y        +
threadIdx.y

out[x][y] = in[y][x]
```



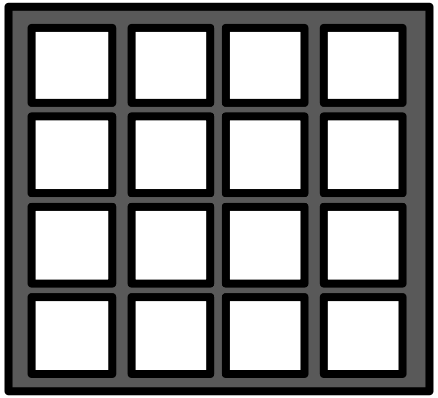


我们可以使用共享内存来实现合并读写。在这里，每个线程块都会分配一个（2,2）共享内存块。

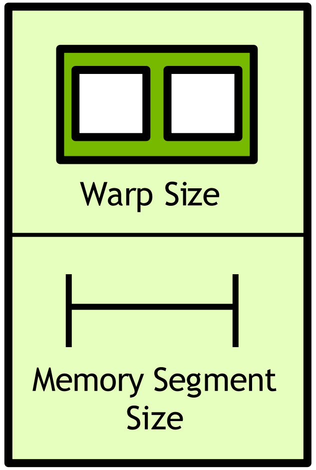
```
tile = cuda.shared.array(2,2)
```



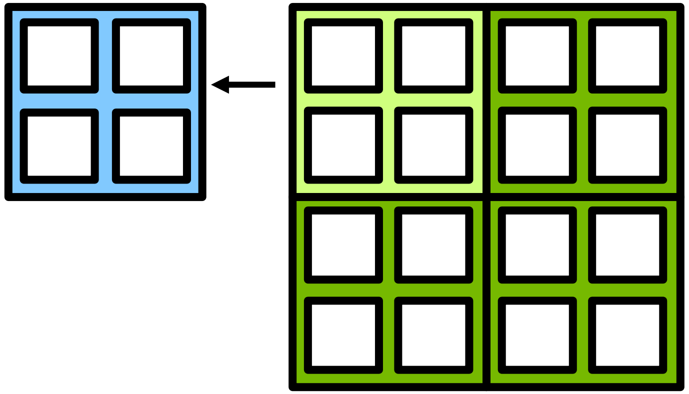
输入



输出

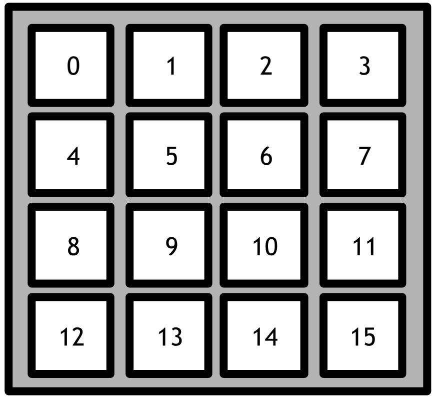


共享内存　网格

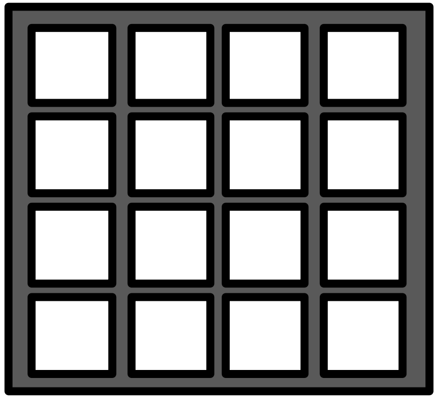


（值得提醒的是，在我们的演示中，为了节省空间，我们设 Warp 的长度为 2。真正的 Warp 有 32 个线程）

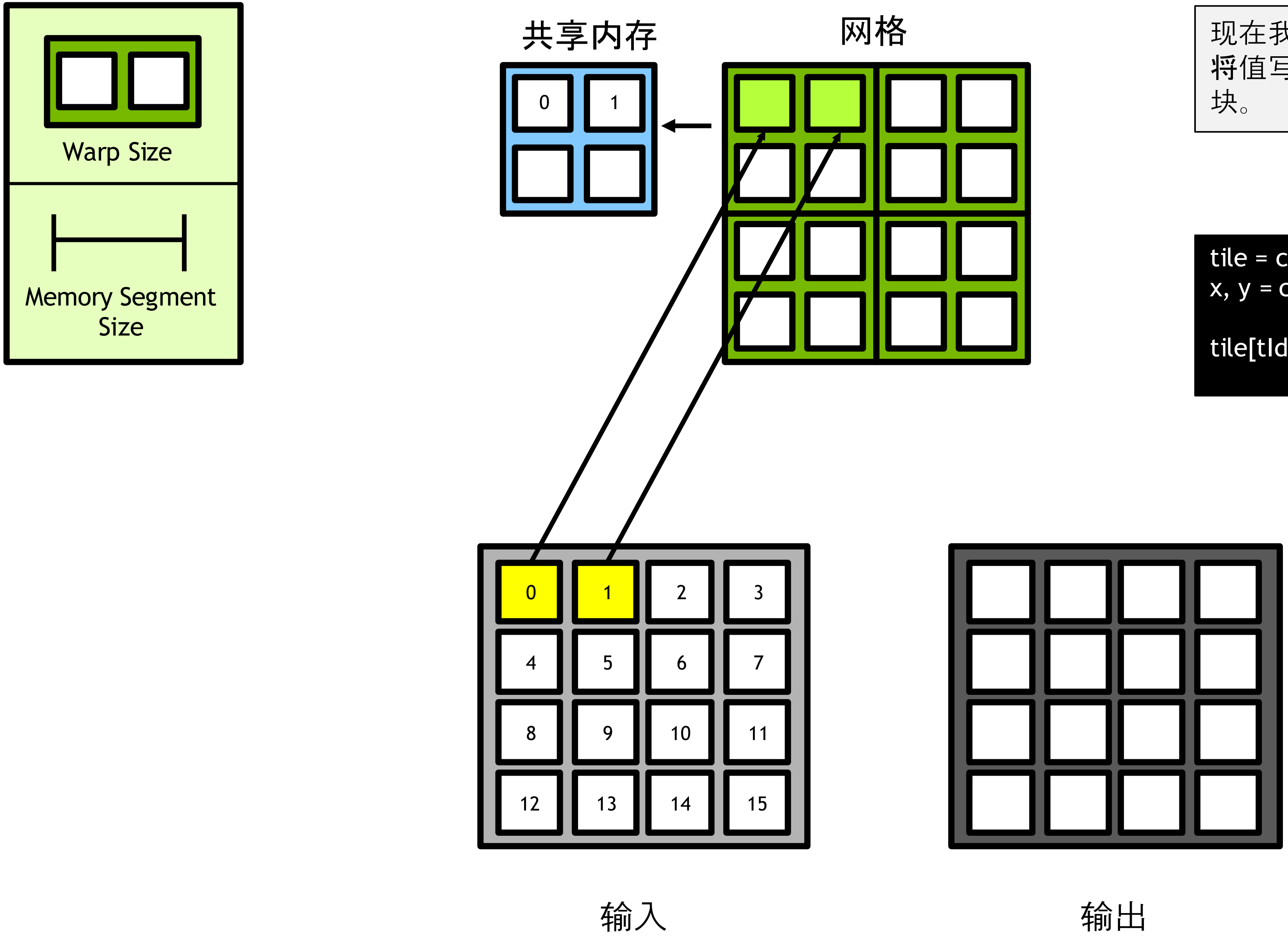
```
tile = cuda.shared.array(2,2)
```



输入

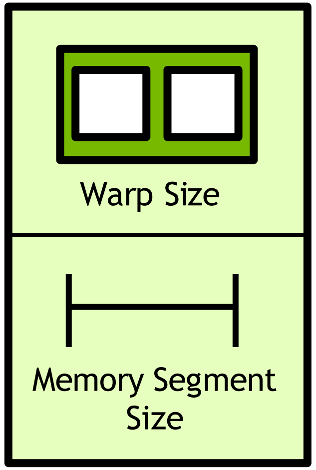


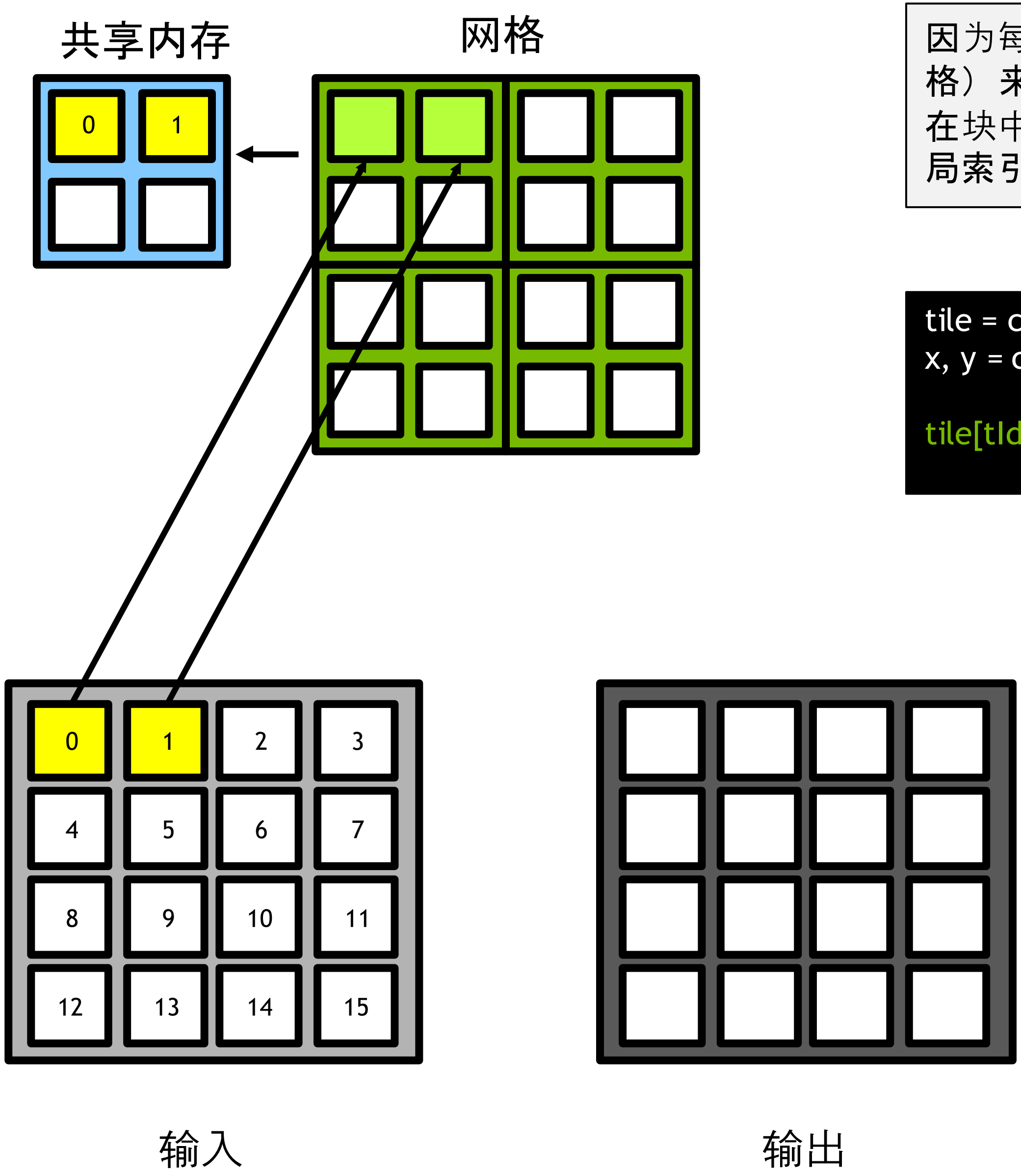
输出



现在我们可以从输入进行合并读取，并将值写入分配给该线程块的的共享内存块。

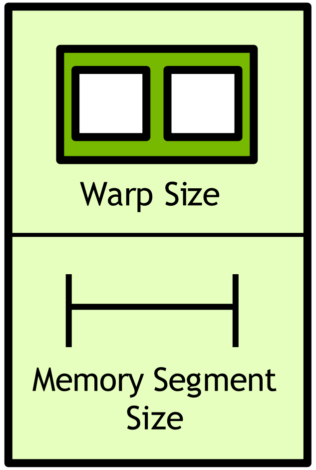
```
tile = cuda.shared.array(2,2)
x, y = cuda.grid(2)

tile[tIdx.y][tIdx.x] = in[y][x]
```

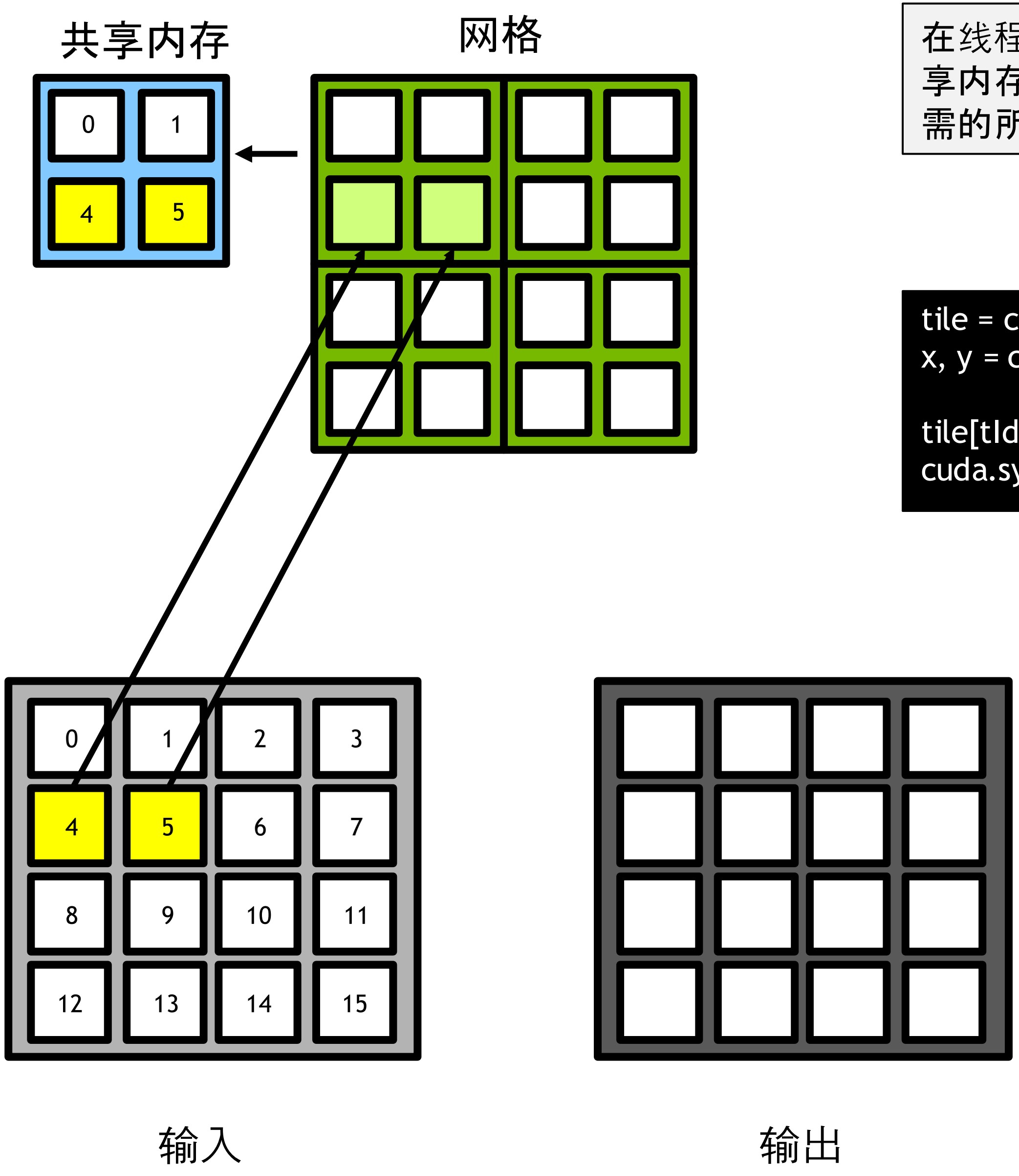




因为每个共享内存块对线程块（不是网格）来说是本地的，所以我们使用线程在块中的本地索引而不是在网格内的全局索引。

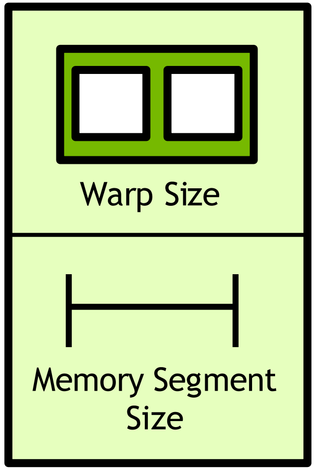
```
tile = cuda.shared.array(2,2)
x, y = cuda.grid(2)

tile[tIdx.y][tIdx.x] = in[y][x]
```

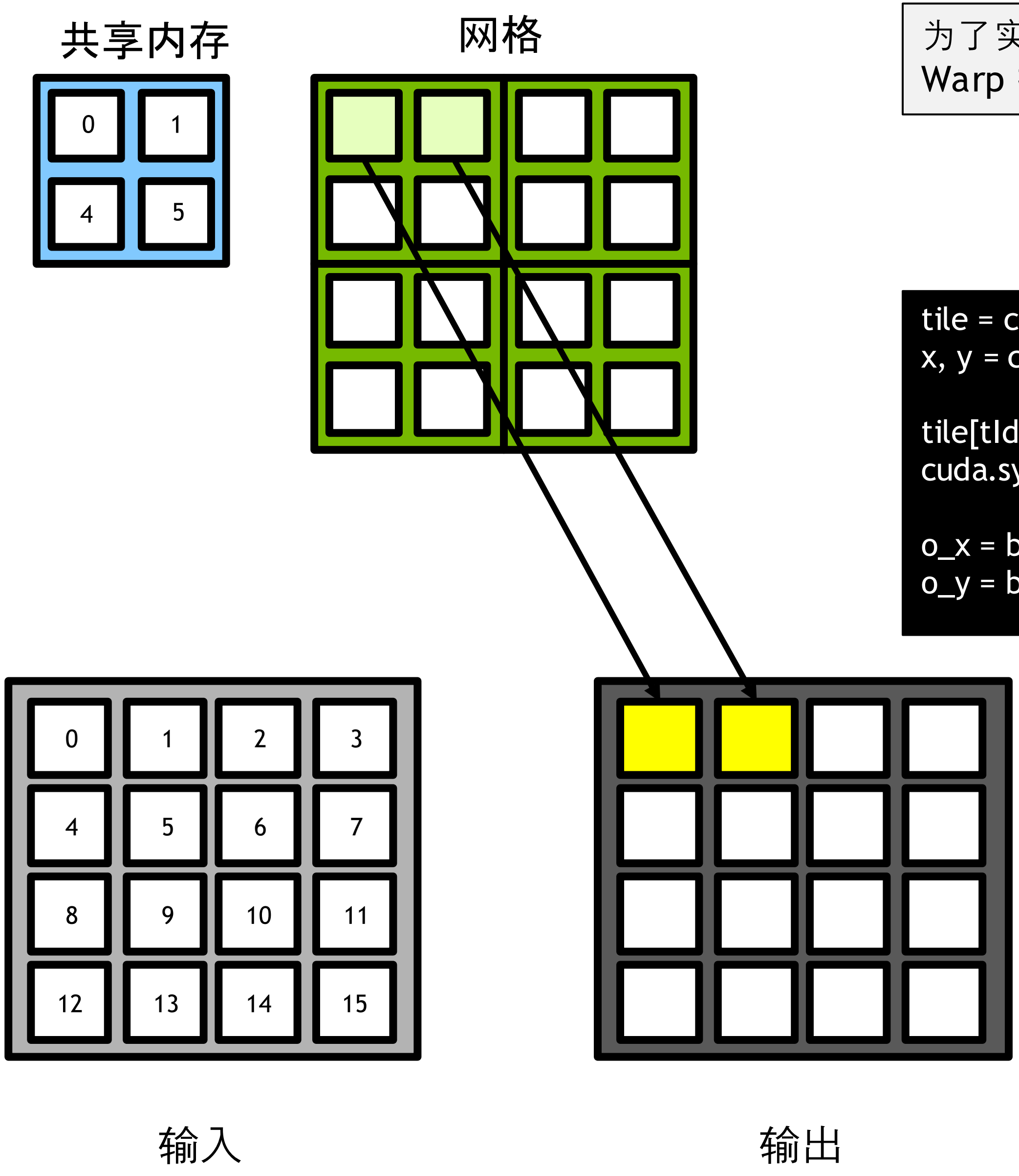




在线程块中的所有线程都同步之后，共享内存块就包含了将要写入输出矩阵所需的所有数据

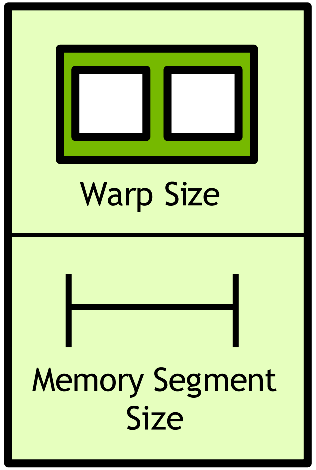
```
tile = cuda.shared.array(2,2)
x, y = cuda.grid(2)

tile[tIdx.y][tIdx.x] = in[y][x]
cuda.syncthreads()
```

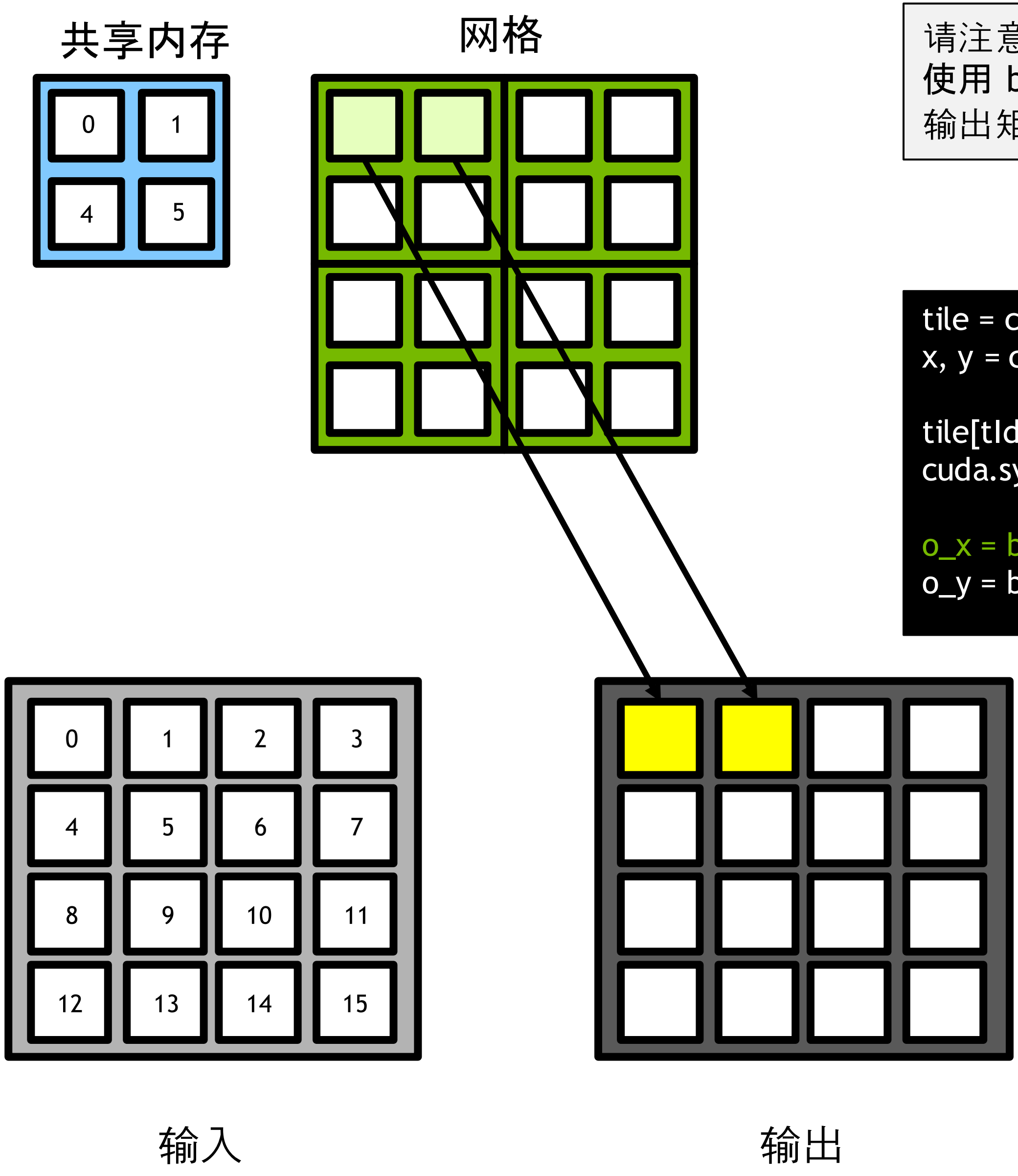




为了实现合并写入，我们希望每个 Warp 按行把数据写入输出矩阵。

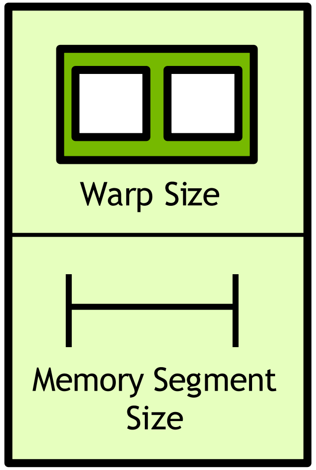
```
tile = cuda.shared.array(2,2)
x, y = cuda.grid(2)

tile[tIdx.y][tIdx.x] = in[y][x]
cuda.syncthreads()

o_x = bId.y*bDim.y + tId.x
o_y = bId.x*bDim.x + tId.y
```

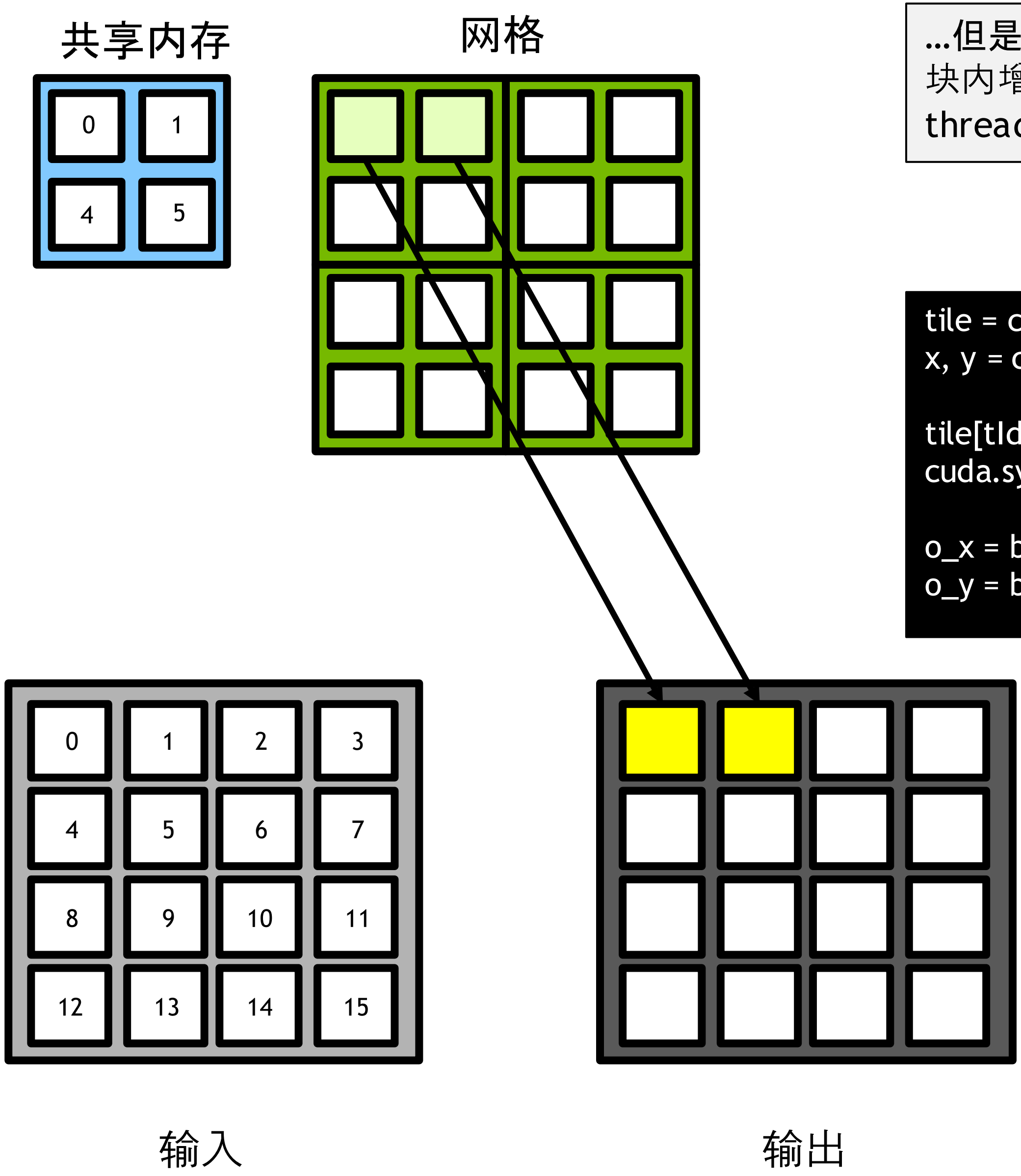




请注意，要在转置位置写入输出，我们使用 blockIdx.y 和 blockDim.y 来计算输出矩阵中的 x 轴索引……

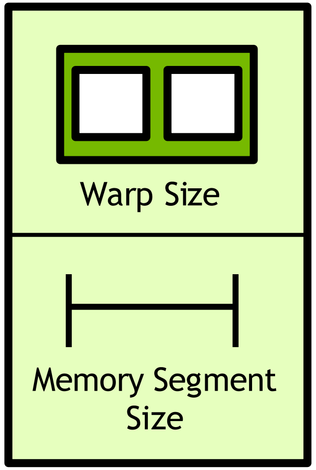
```
tile = cuda.shared.array(2,2)
x, y = cuda.grid(2)

tile[tIdx.y][tIdx.x] = in[y][x]
cuda.syncthreads()

o_x = bId.y*bDim.y + tId.x
o_y = bId.x*bDim.x + tId.y
```

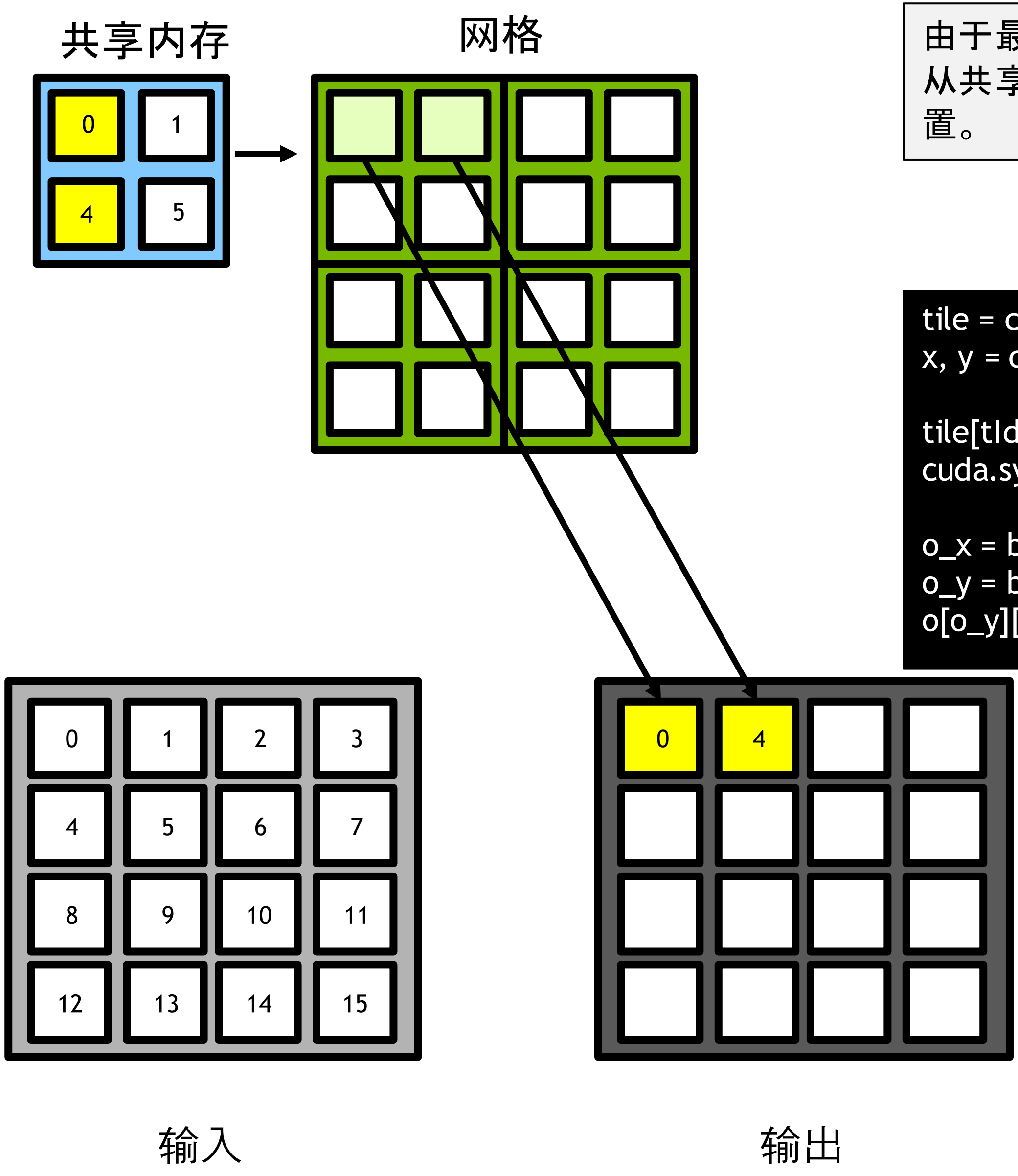




...但是为了完成合并写入，我们仍然将块内增量沿着 x 输出轴映射到 threadIdx.x 。

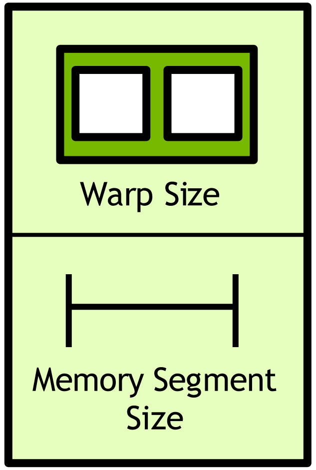
```
tile = cuda.shared.array(2,2)
x, y = cuda.grid(2)

tile[tIdx.y][tIdx.x] = in[y][x]
cuda.syncthreads()

o_x = bId.y*bDim.y + tId.x
o_y = bId.x*bDim.x + tId.y
```

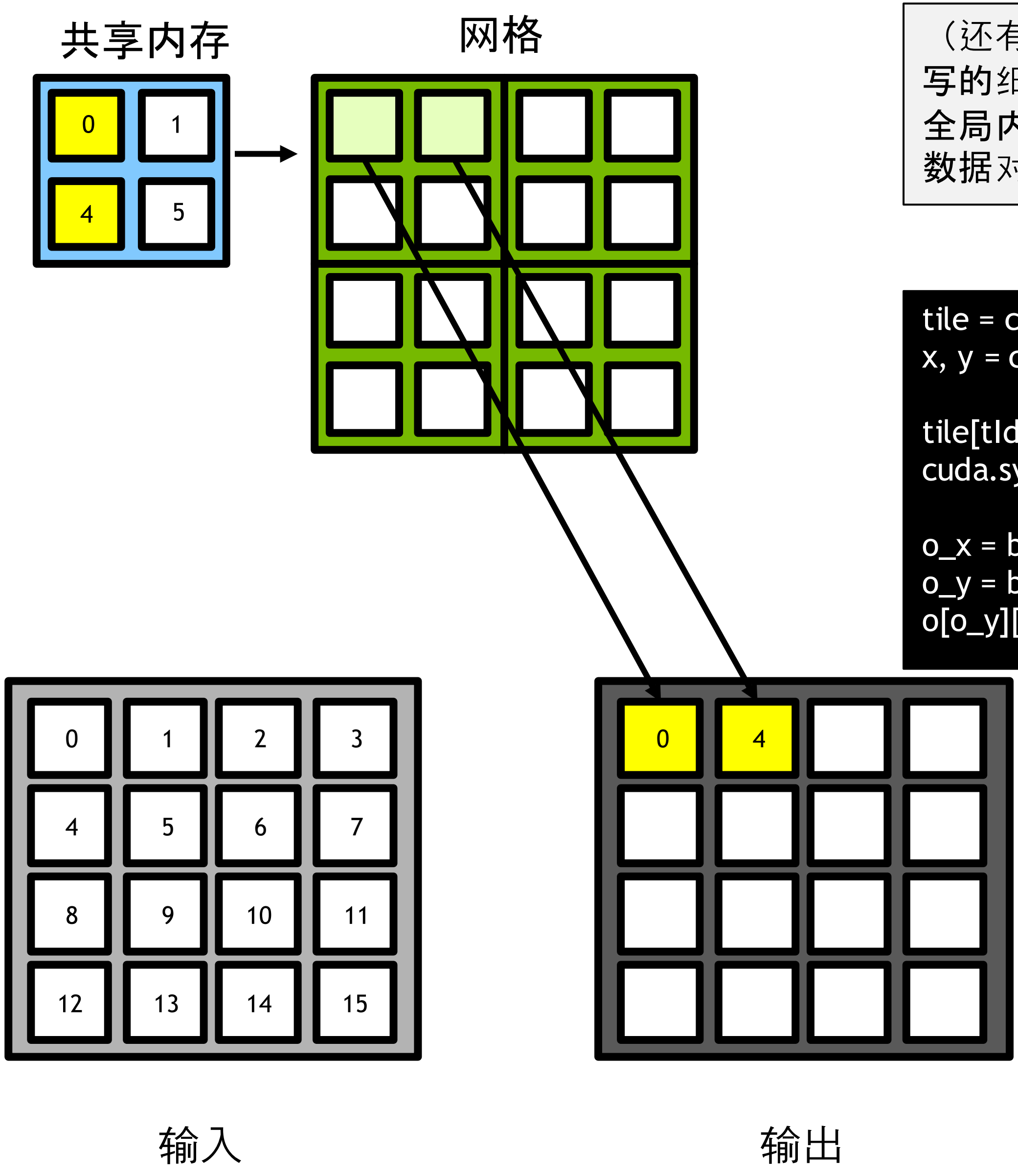




由于最后这个细节，每个 Warp 都需要从共享内存块的列中读取数据以执行转置。

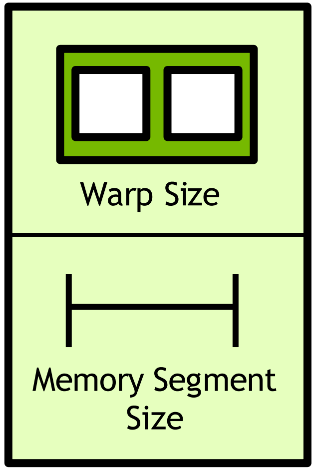
```
tile = cuda.shared.array(2,2)
x, y = cuda.grid(2)

tile[tIdx.y][tIdx.x] = in[y][x]
cuda.syncthreads()

o_x = bId.y*bDim.y + tId.x
o_y = bId.x*bDim.x + tId.y
o[o_y][o_x] = tile[tIdx.x][tIdx.y]
```



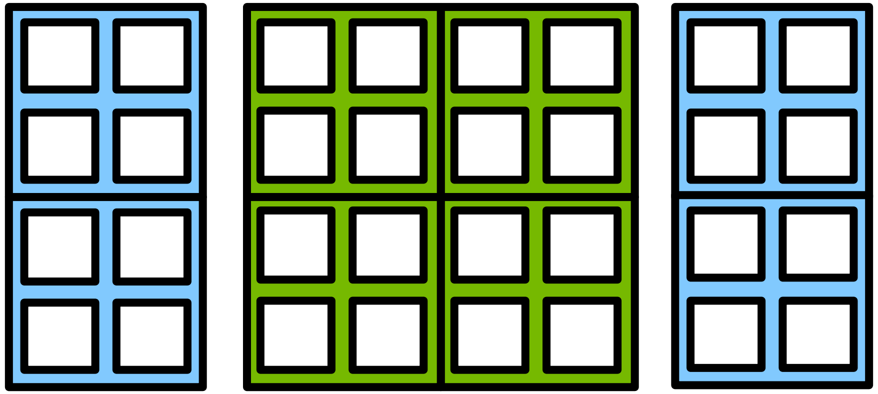


（还有更多关于对共享内存的高效读/写的细节，但现在您只需知道，与使用全局内存相比，在共享内存中跨列读取数据对性能的影响非常小）

```
tile = cuda.shared.array(2,2)
x, y = cuda.grid(2)

tile[tIdx.y][tIdx.x] = in[y][x]
cuda.syncthreads()

o_x = bId.y*bDim.y + tId.x
o_y = bId.x*bDim.x + tId.y
o[o_y][o_x] = tile[tIdx.x][tIdx.y]
```

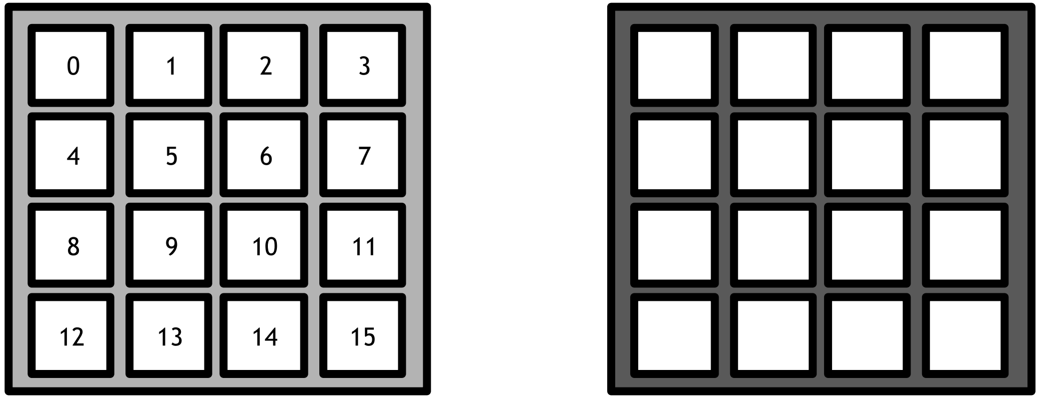


共享内存　　网格　　共享内存

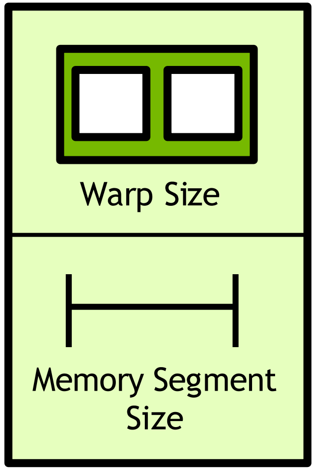


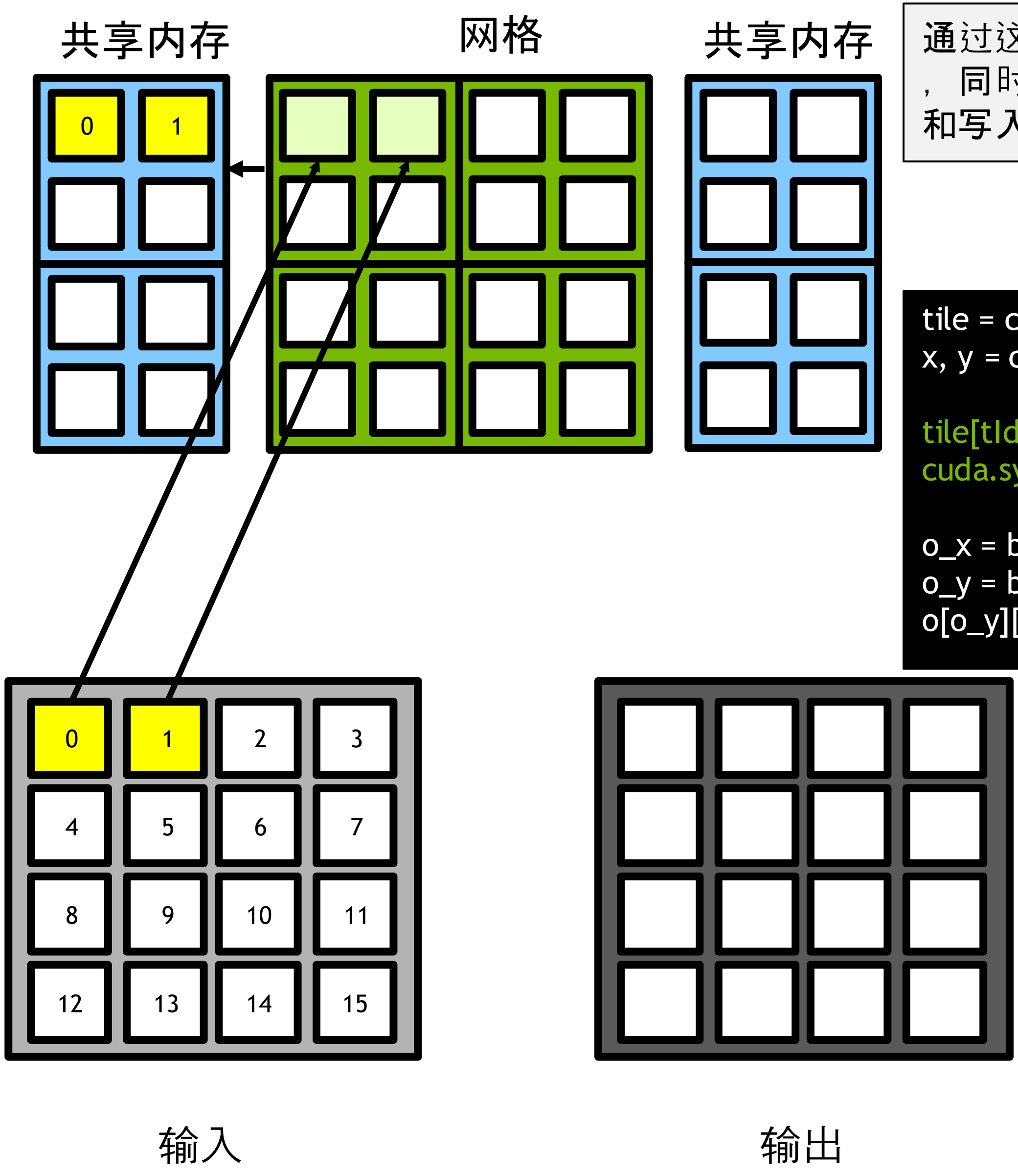
通过这种方式，我们可以实现矩阵转置，同时对全局内存进行完全合并的读取和写入。

```
tile = cuda.shared.array(2,2)
x, y = cuda.grid(2)

tile[tIdx.y][tIdx.x] = in[y][x]
cuda.syncthreads()

o_x = bId.y*bDim.y + tId.x
o_y = bId.x*bDim.x + tId.y
o[o_y][o_x] = tile[tIdx.x][tIdx.y]
```



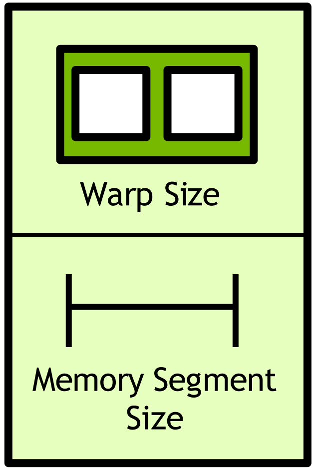
输入　　输出

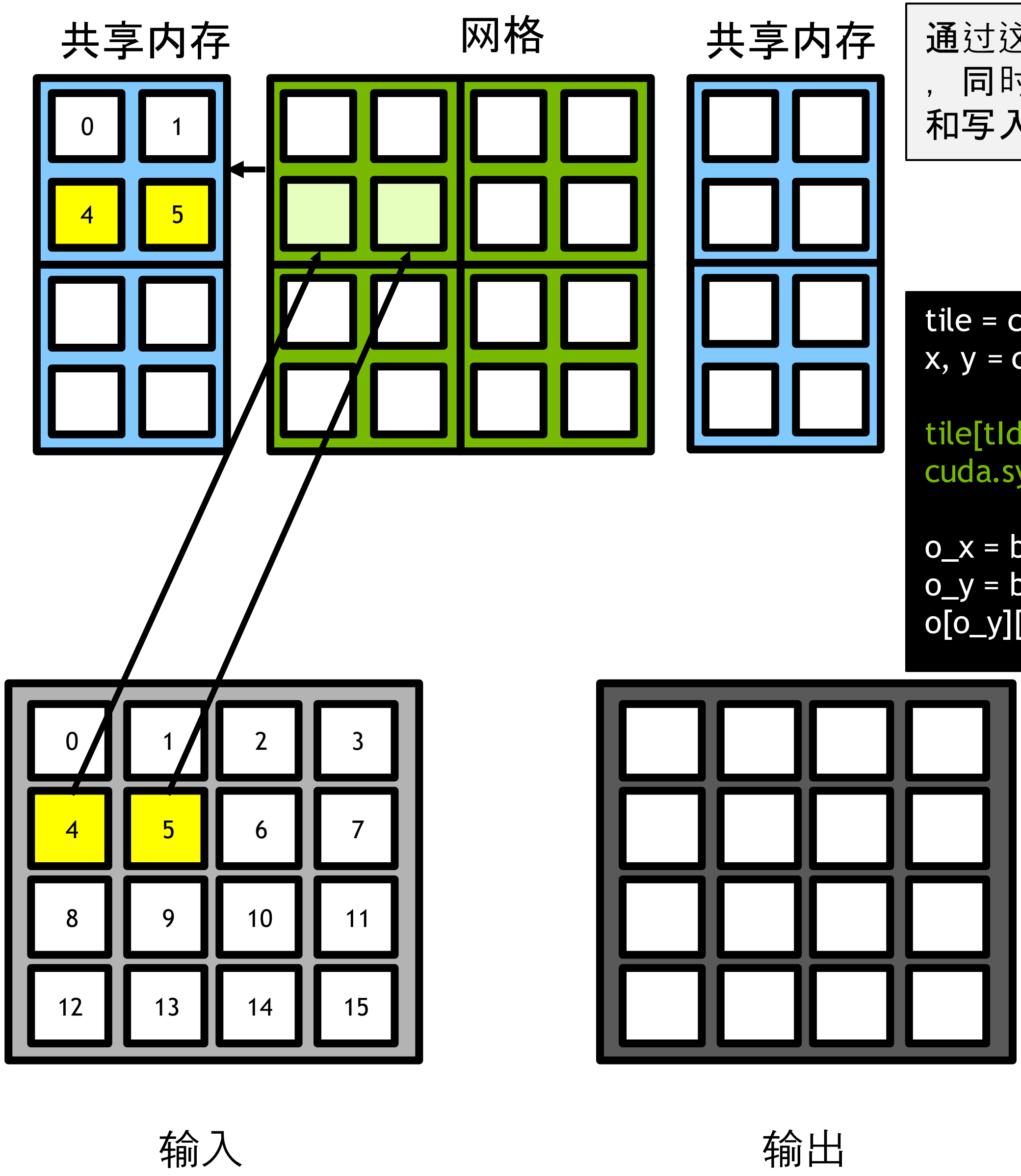




通过这种方式，我们可以实现矩阵转置，同时对全局内存进行完全合并的读取和写入。

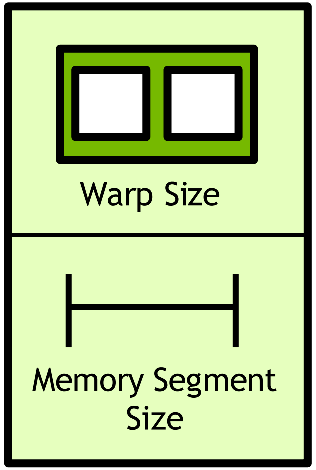
```
tile = cuda.shared.array(2,2)
x, y = cuda.grid(2)

tile[tIdx.y][tIdx.x] = in[y][x]
cuda.syncthreads()

o_x = bId.y*bDim.y + tId.x
o_y = bId.x*bDim.x + tId.y
o[o_y][o_x] = tile[tIdx.x][tIdx.y]
```

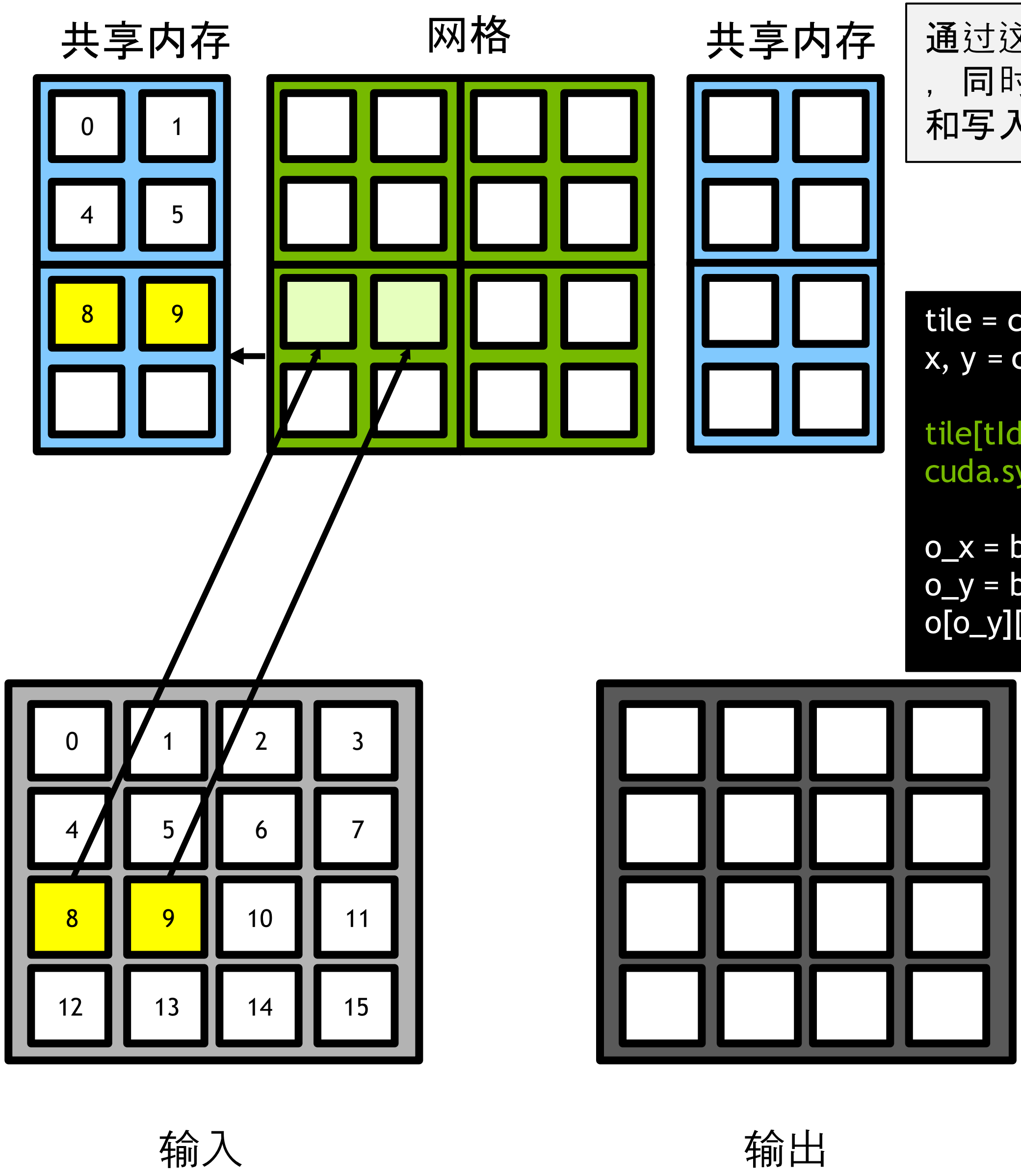




通过这种方式，我们可以实现矩阵转置，同时对全局内存进行完全合并的读取和写入。

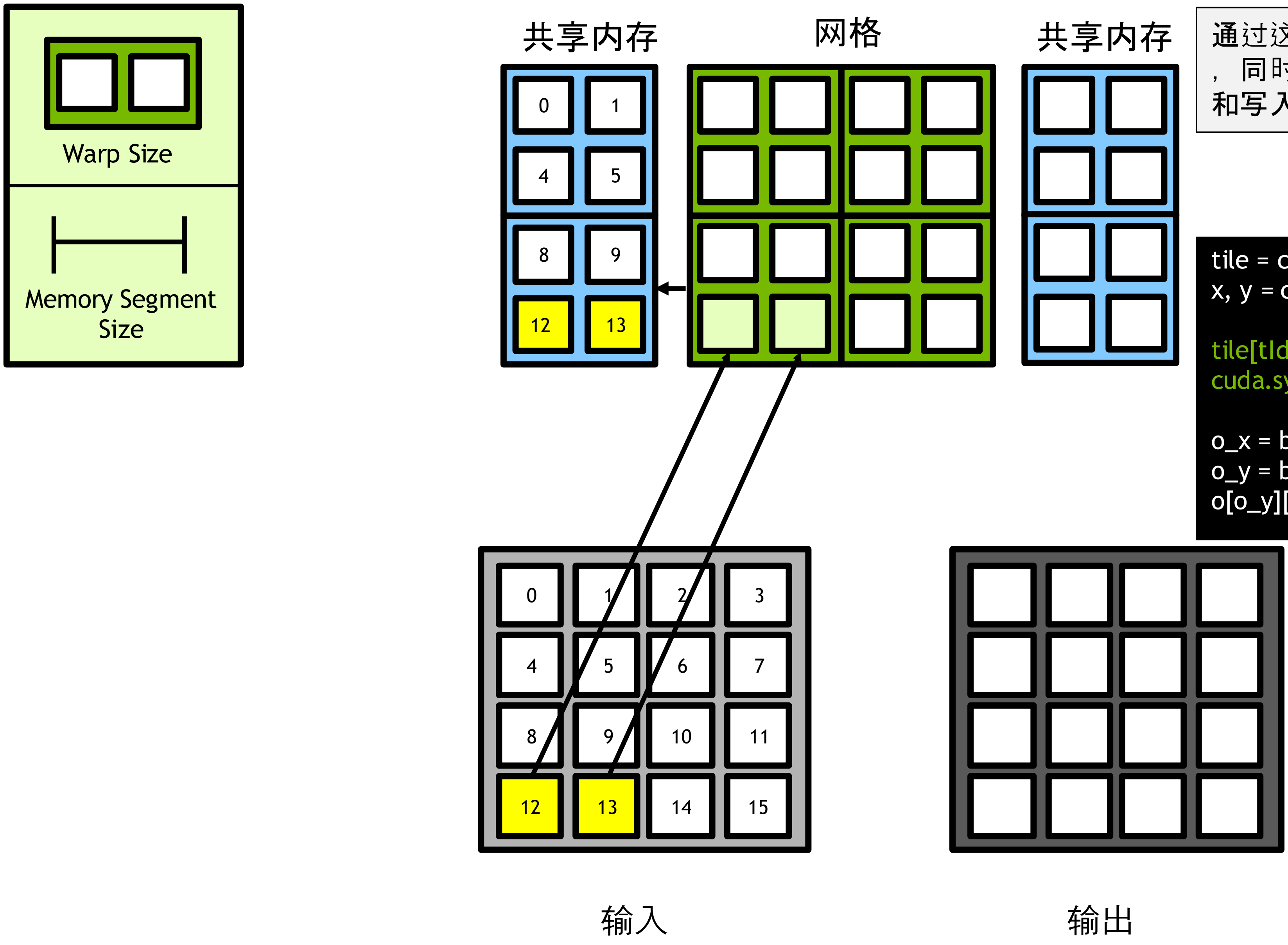
```
tile = cuda.shared.array(2,2)
x, y = cuda.grid(2)

tile[tIdx.y][tIdx.x] = in[y][x]
cuda.syncthreads()

o_x = bId.y*bDim.y + tId.x
o_y = bId.x*bDim.x + tId.y
o[o_y][o_x] = tile[tIdx.x][tIdx.y]
```





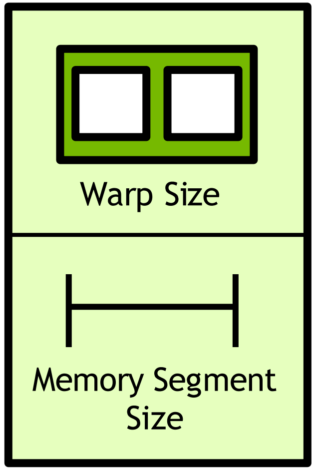
通过这种方式，我们可以实现矩阵转置，同时对全局内存进行完全合并的读取和写入。

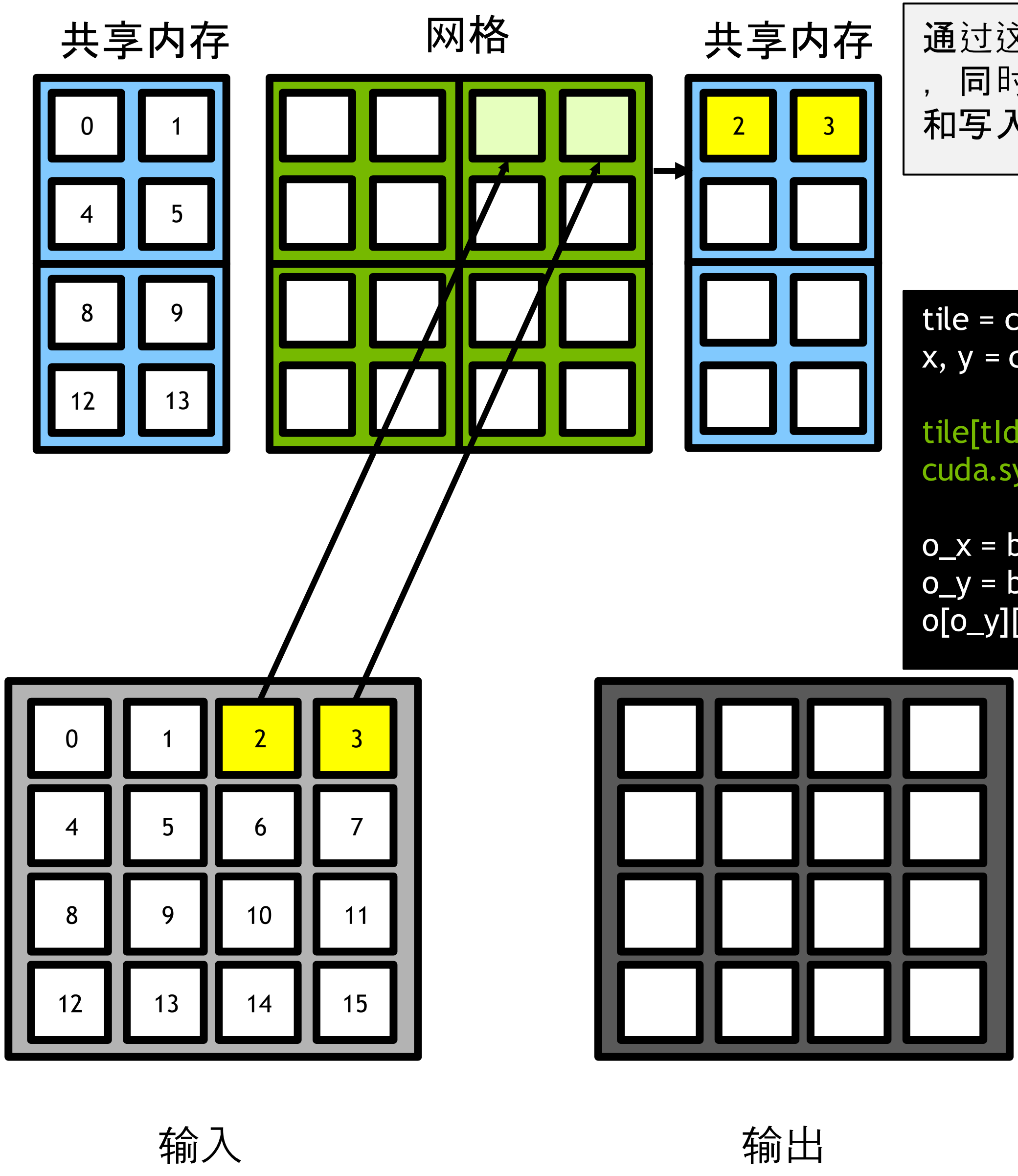
```
tile = cuda.shared.array(2,2)
x, y = cuda.grid(2)

tile[tIdx.y][tIdx.x] = in[y][x]
cuda.syncthreads()

o_x = bId.y*bDim.y + tId.x
o_y = bId.x*bDim.x + tId.y
o[o_y][o_x] = tile[tIdx.x][tIdx.y]
```



通过这种方式，我们可以实现矩阵转置，同时对全局内存进行完全合并的读取和写入。

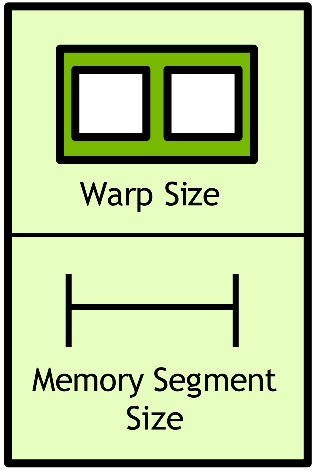
```
tile = cuda.shared.array(2,2)
x, y = cuda.grid(2)

tile[tIdx.y][tIdx.x] = in[y][x]
cuda.syncthreads()

o_x = bId.y*bDim.y + tId.x
o_y = bId.x*bDim.x + tId.y
o[o_y][o_x] = tile[tIdx.x][tIdx.y]
```

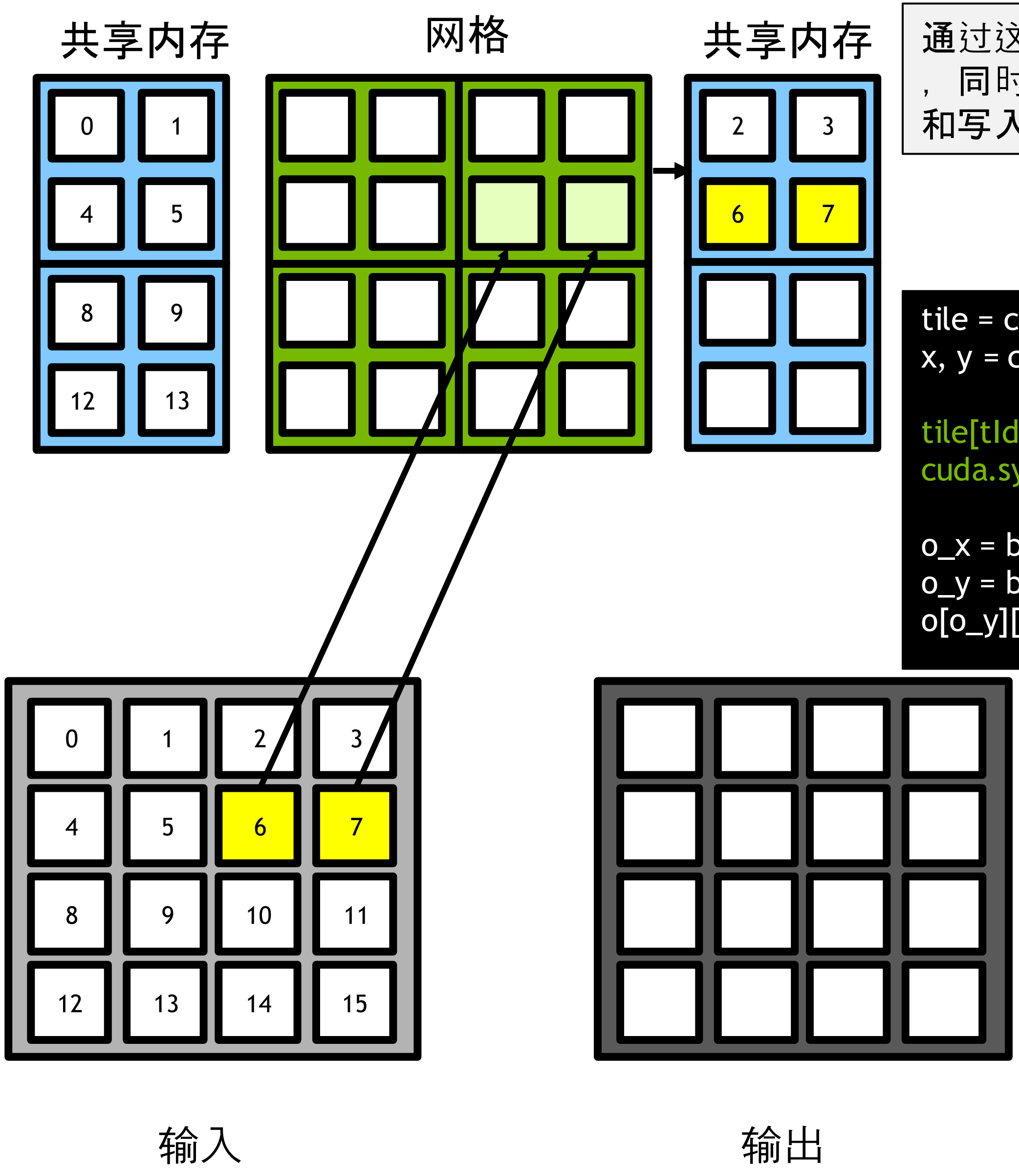




通过这种方式，我们可以实现矩阵转置，同时对全局内存进行完全合并的读取和写入。

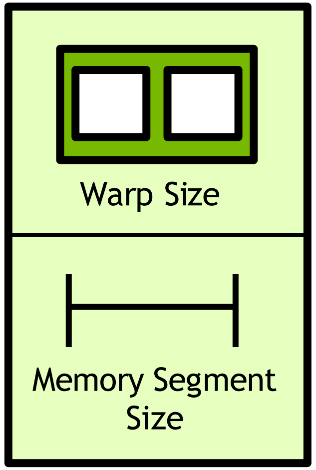
```
tile = cuda.shared.array(2,2)
x, y = cuda.grid(2)

tile[tIdx.y][tIdx.x] = in[y][x]
cuda.syncthreads()

o_x = bId.y*bDim.y + tId.x
o_y = bId.x*bDim.x + tId.y
o[o_y][o_x] = tile[tIdx.x][tIdx.y]
```

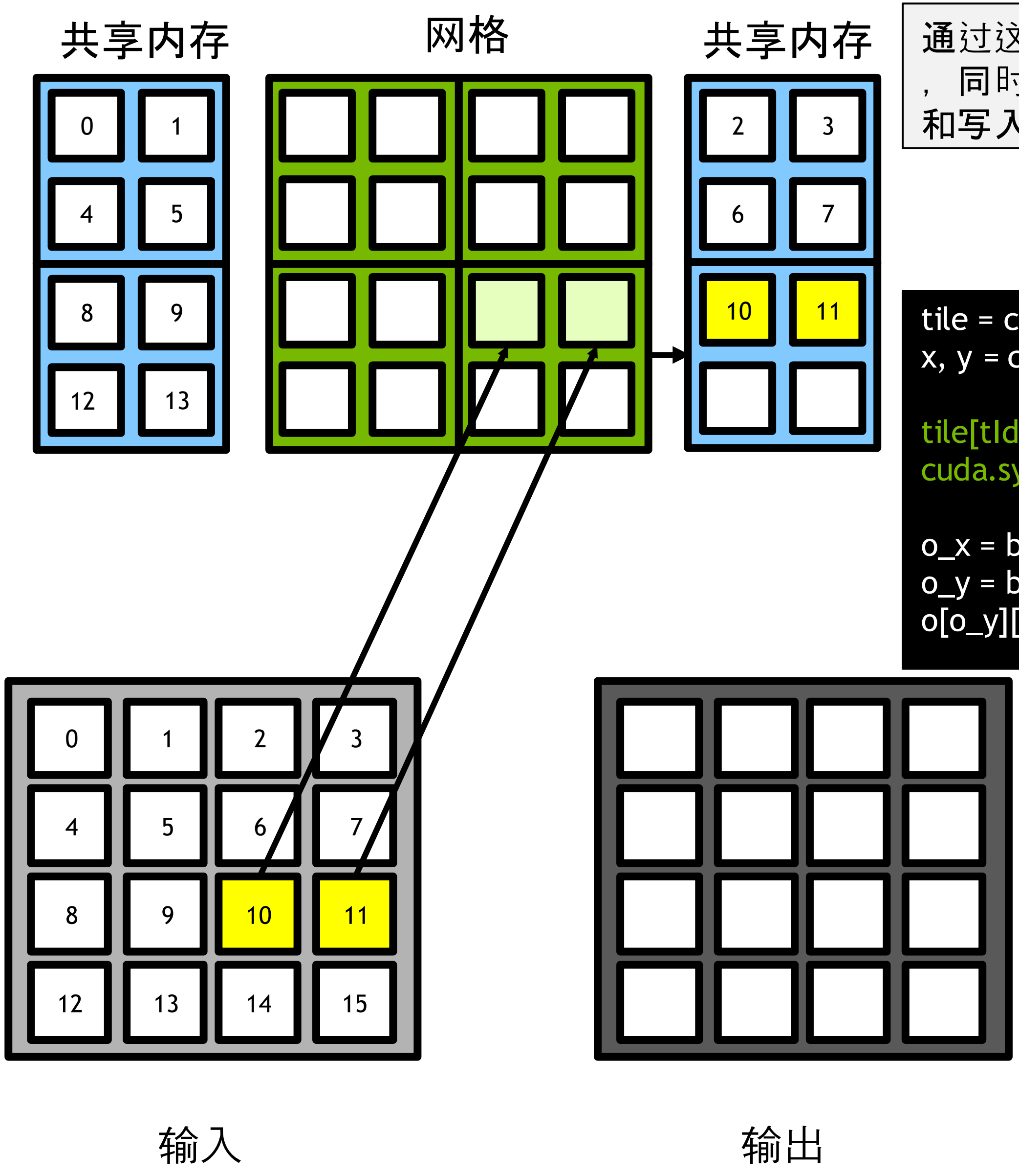




通过这种方式，我们可以实现矩阵转置，同时对全局内存进行完全合并的读取和写入。

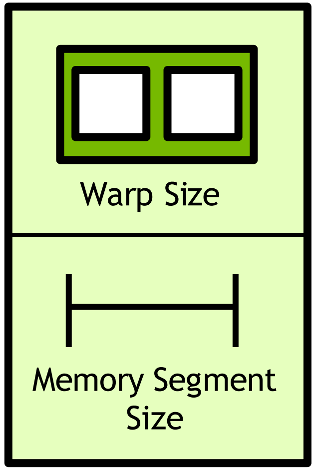
```
tile = cuda.shared.array(2,2)
x, y = cuda.grid(2)

tile[tIdx.y][tIdx.x] = in[y][x]
cuda.syncthreads()

o_x = bId.y*bDim.y + tId.x
o_y = bId.x*bDim.x + tId.y
o[o_y][o_x] = tile[tIdx.x][tIdx.y]
```

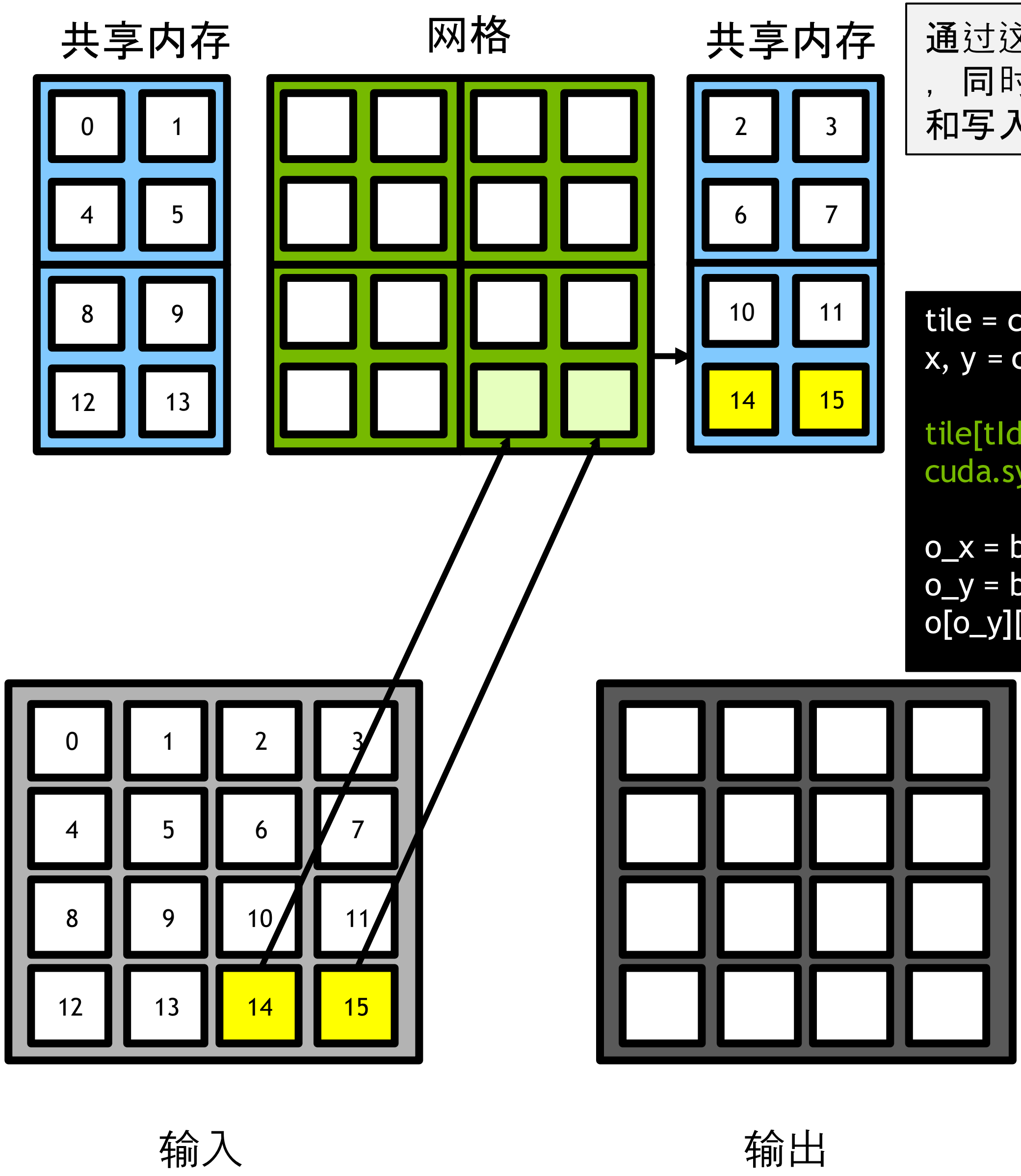




通过这种方式，我们可以实现矩阵转置，同时对全局内存进行完全合并的读取和写入。

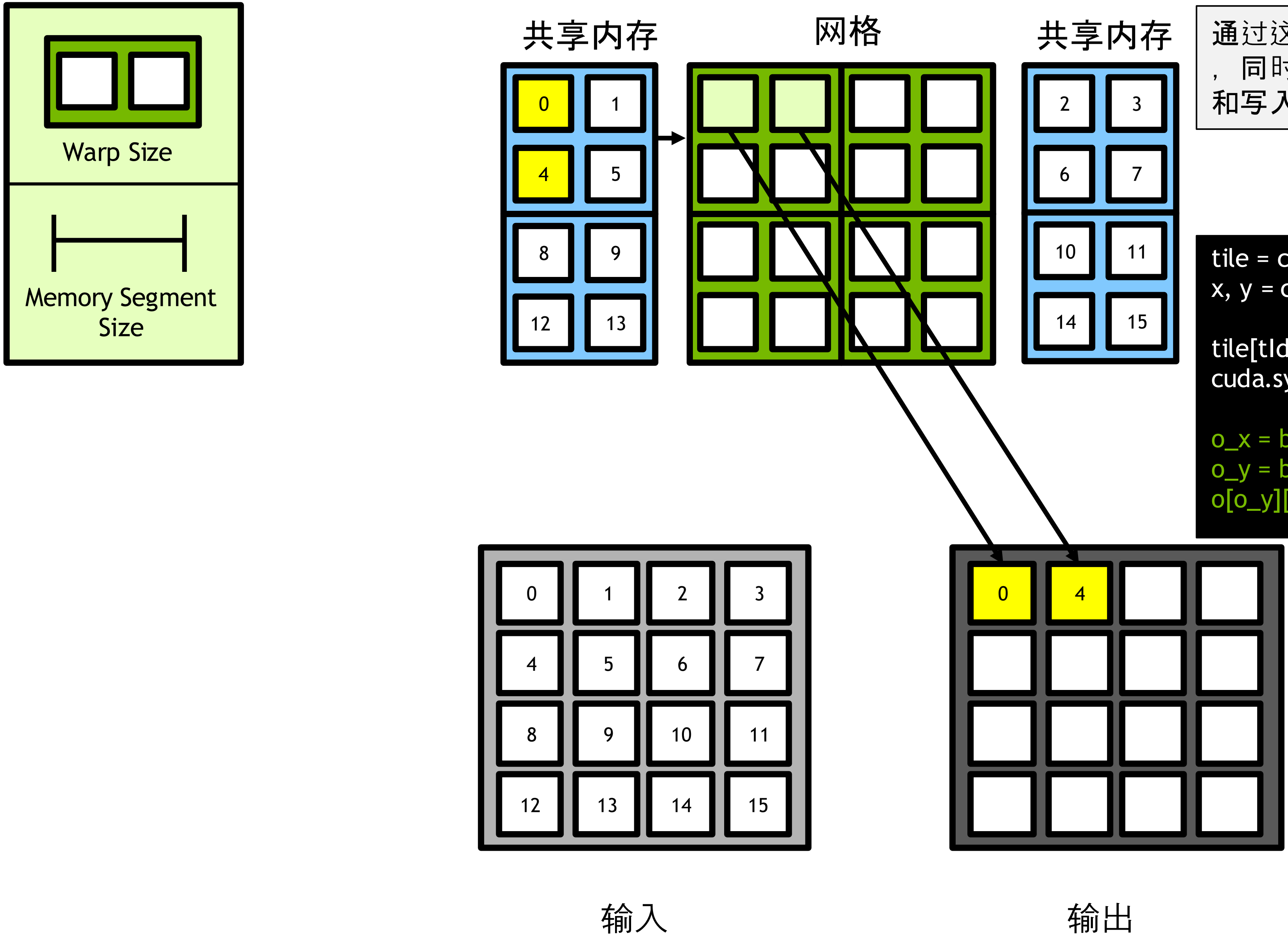
```
tile = cuda.shared.array(2,2)
x, y = cuda.grid(2)

tile[tIdx.y][tIdx.x] = in[y][x]
cuda.syncthreads()

o_x = bId.y*bDim.y + tId.x
o_y = bId.x*bDim.x + tId.y
o[o_y][o_x] = tile[tIdx.x][tIdx.y]
```





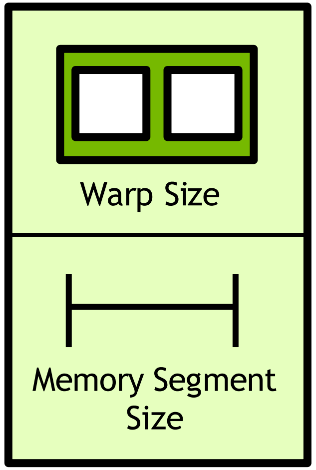
通过这种方式，我们可以实现矩阵转置，同时对全局内存进行完全合并的读取和写入。

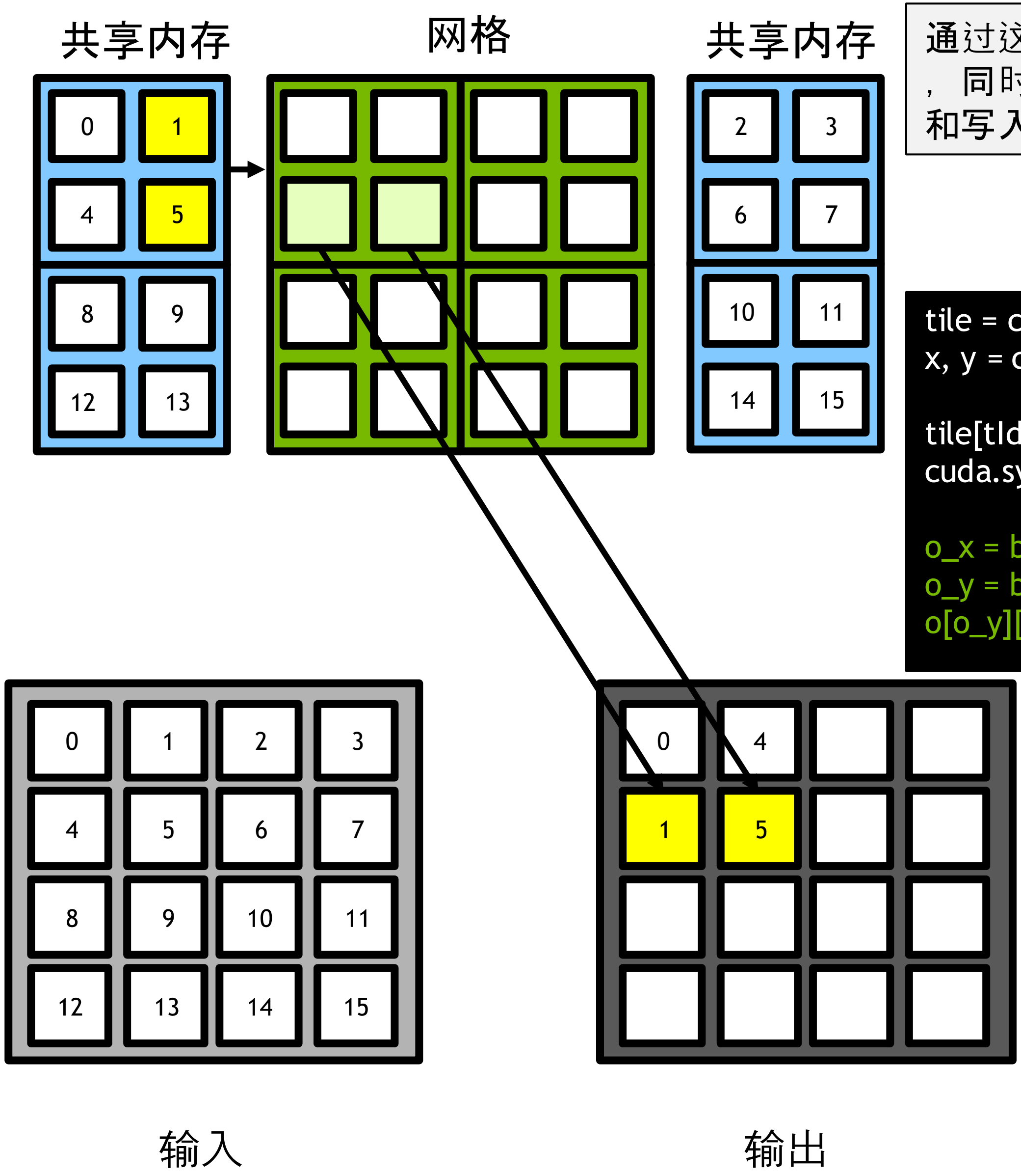
```
tile = cuda.shared.array(2,2)
x, y = cuda.grid(2)

tile[tIdx.y][tIdx.x] = in[y][x]
cuda.syncthreads()

o_x = bId.y*bDim.y + tId.x
o_y = bId.x*bDim.x + tId.y
o[o_y][o_x] = tile[tIdx.x][tIdx.y]
```



通过这种方式，我们可以实现矩阵转置，同时对全局内存进行完全合并的读取和写入。

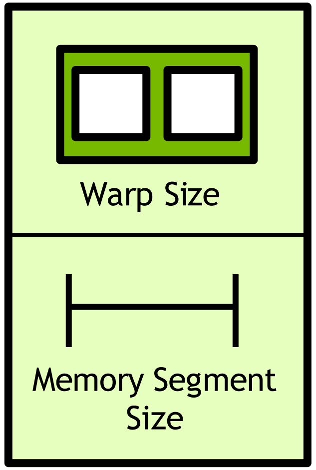
```
tile = cuda.shared.array(2,2)
x, y = cuda.grid(2)

tile[tIdx.y][tIdx.x] = in[y][x]
cuda.syncthreads()

o_x = bId.y*bDim.y + tId.x
o_y = bId.x*bDim.x + tId.y
o[o_y][o_x] = tile[tIdx.x][tIdx.y]
```

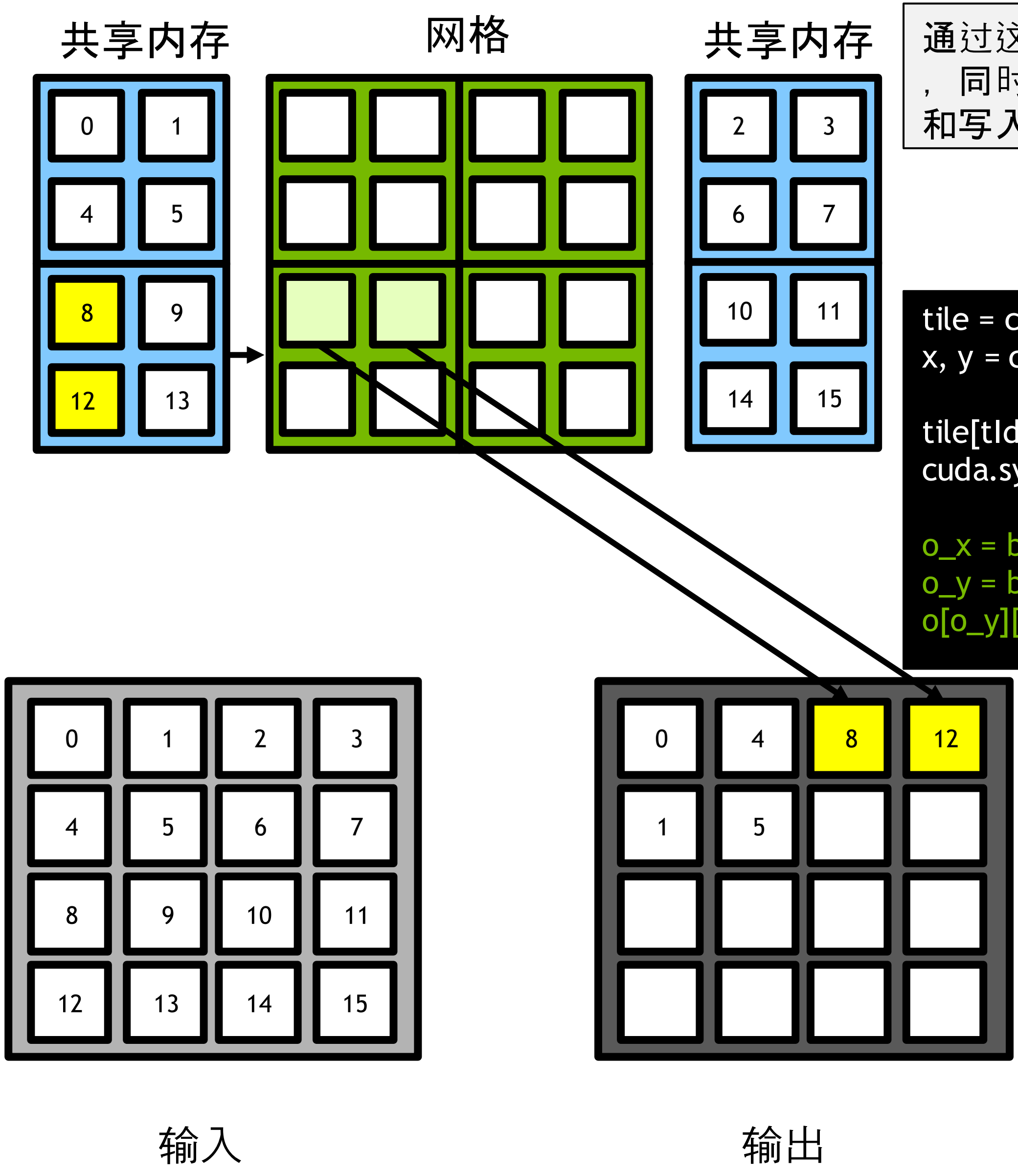




通过这种方式，我们可以实现矩阵转置，同时对全局内存进行完全合并的读取和写入。

```
tile = cuda.shared.array(2,2)
x, y = cuda.grid(2)

tile[tIdx.y][tIdx.x] = in[y][x]
cuda.syncthreads()

o_x = bId.y*bDim.y + tId.x
o_y = bId.x*bDim.x + tId.y
o[o_y][o_x] = tile[tIdx.x][tIdx.y]
```

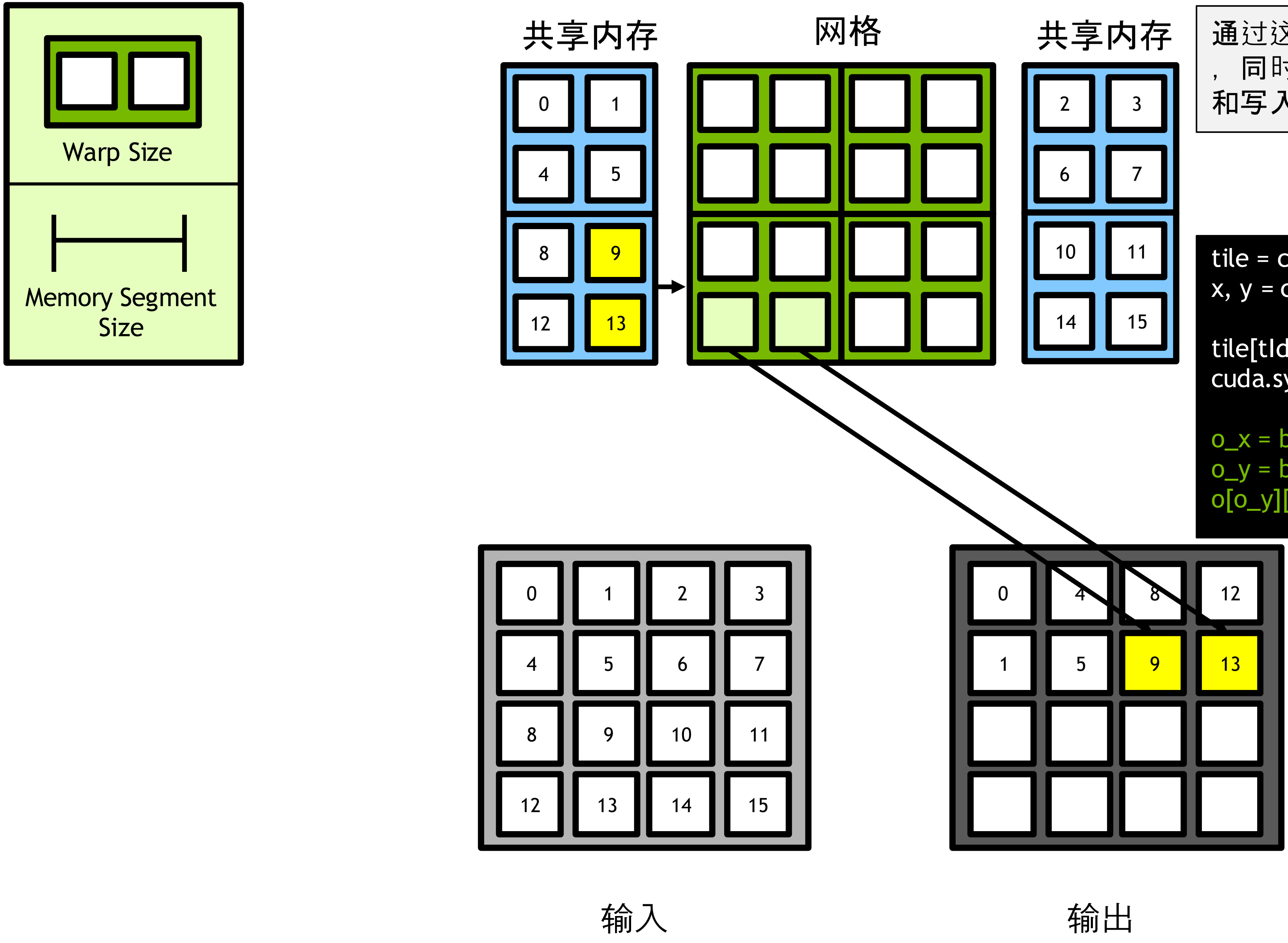




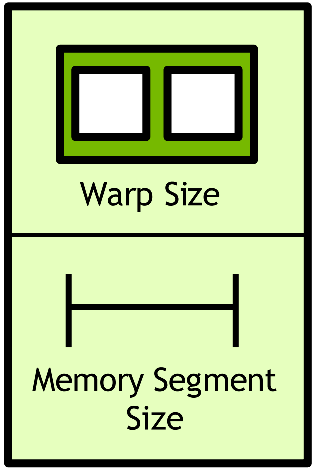
通过这种方式，我们可以实现矩阵转置，同时对全局内存进行完全合并的读取和写入。

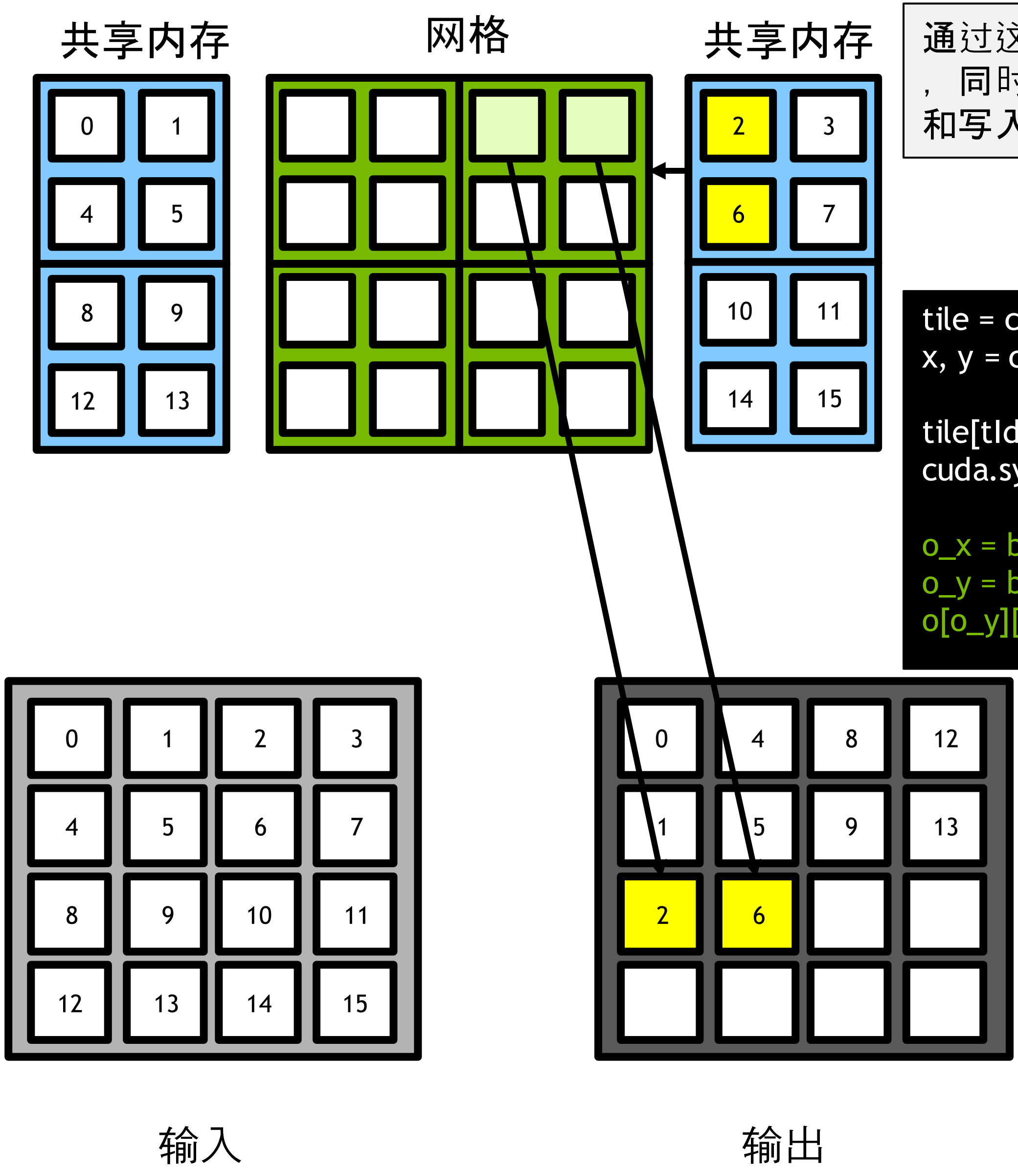
```
tile = cuda.shared.array(2,2)
x, y = cuda.grid(2)

tile[tIdx.y][tIdx.x] = in[y][x]
cuda.syncthreads()

o_x = bId.y*bDim.y + tId.x
o_y = bId.x*bDim.x + tId.y
o[o_y][o_x] = tile[tIdx.x][tIdx.y]
```

通过这种方式，我们可以实现矩阵转置，同时对全局内存进行完全合并的读取和写入。



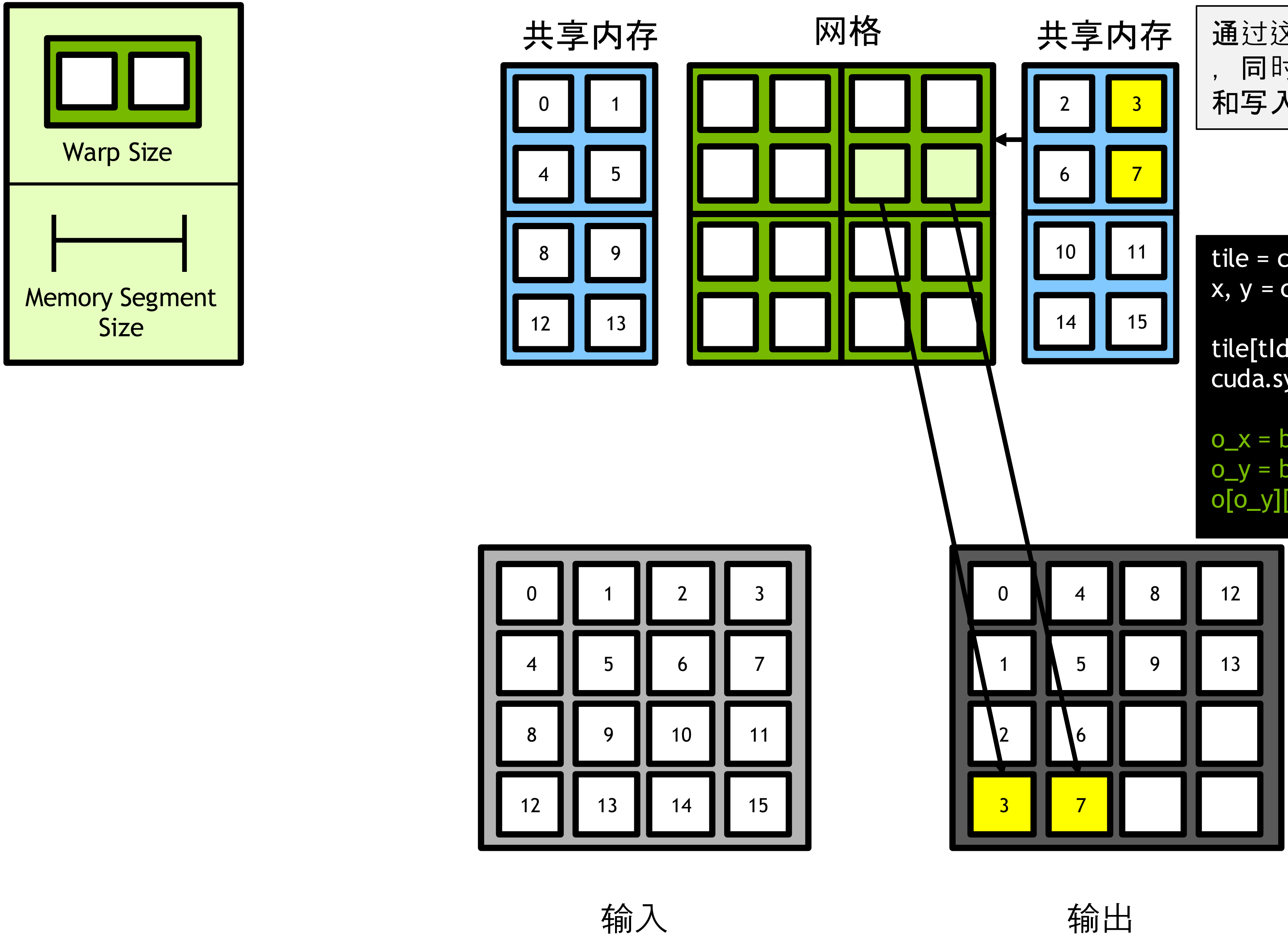

```
tile = cuda.shared.array(2,2)
x, y = cuda.grid(2)

tile[tIdx.y][tIdx.x] = in[y][x]
cuda.syncthreads()

o_x = bId.y*bDim.y + tId.x
o_y = bId.x*bDim.x + tId.y
o[o_y][o_x] = tile[tIdx.x][tIdx.y]
```





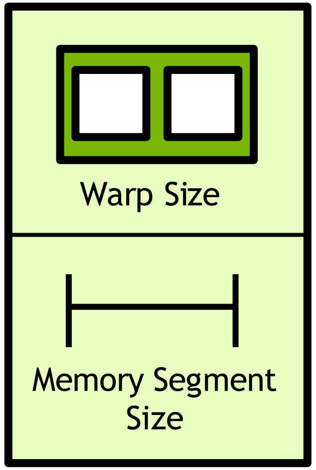
通过这种方式，我们可以实现矩阵转置，同时对全局内存进行完全合并的读取和写入。

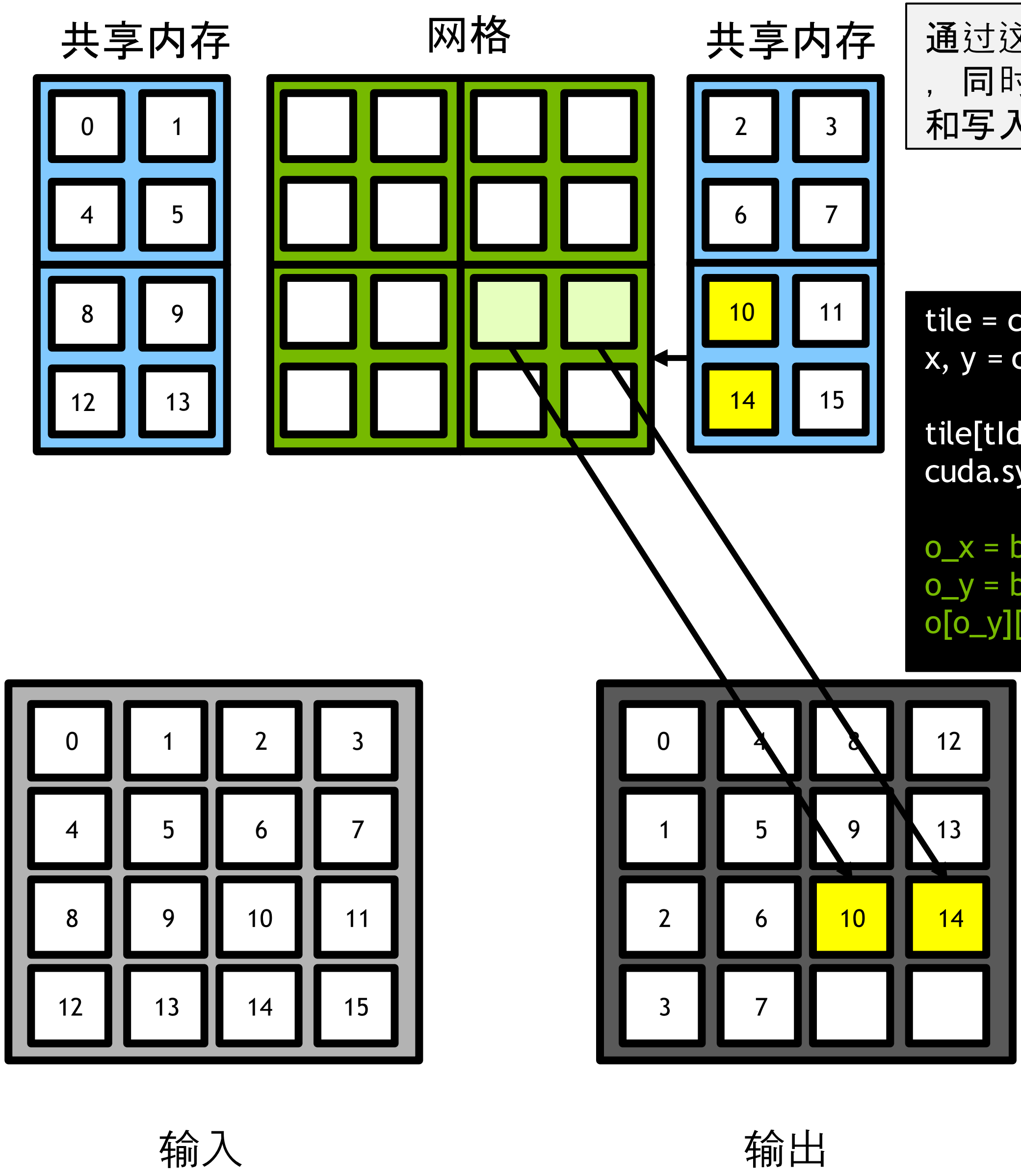
```
tile = cuda.shared.array(2,2)
x, y = cuda.grid(2)

tile[tIdx.y][tIdx.x] = in[y][x]
cuda.syncthreads()

o_x = bId.y*bDim.y + tId.x
o_y = bId.x*bDim.x + tId.y
o[o_y][o_x] = tile[tIdx.x][tIdx.y]
```



通过这种方式，我们可以实现矩阵转置，同时对全局内存进行完全合并的读取和写入。

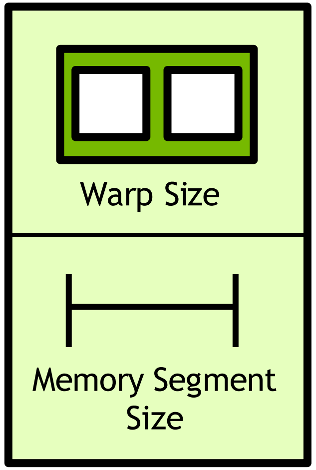
```
tile = cuda.shared.array(2,2)
x, y = cuda.grid(2)

tile[tIdx.y][tIdx.x] = in[y][x]
cuda.syncthreads()

o_x = bId.y*bDim.y + tId.x
o_y = bId.x*bDim.x + tId.y
o[o_y][o_x] = tile[tIdx.x][tIdx.y]
```

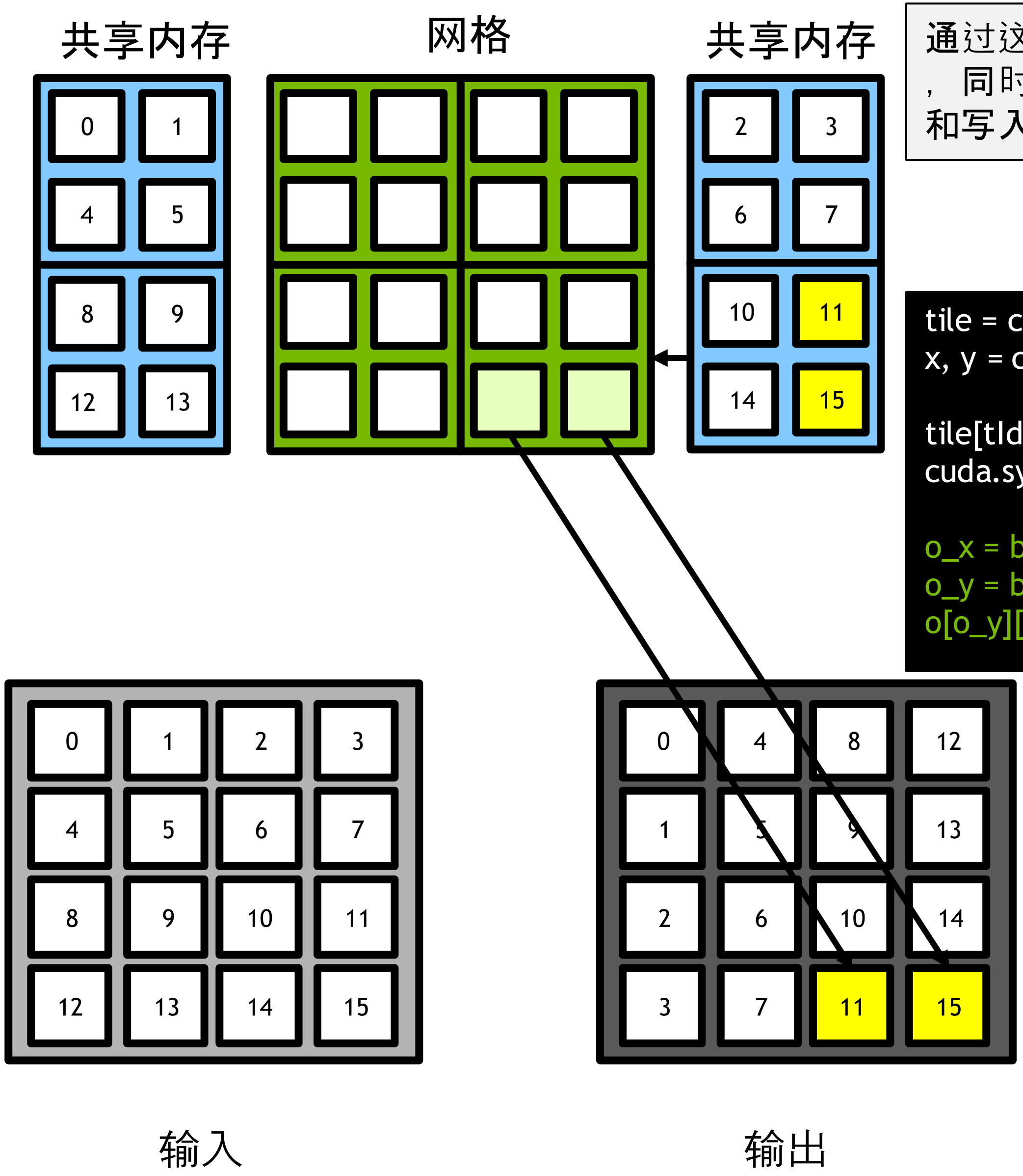




通过这种方式，我们可以实现矩阵转置，同时对全局内存进行完全合并的读取和写入。

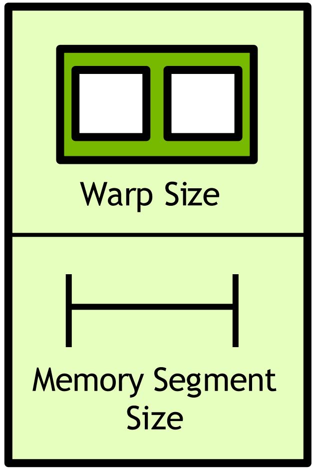
```
tile = cuda.shared.array(2,2)
x, y = cuda.grid(2)

tile[tIdx.y][tIdx.x] = in[y][x]
cuda.syncthreads()

o_x = bId.y*bDim.y + tId.x
o_y = bId.x*bDim.x + tId.y
o[o_y][o_x] = tile[tIdx.x][tIdx.y]
```

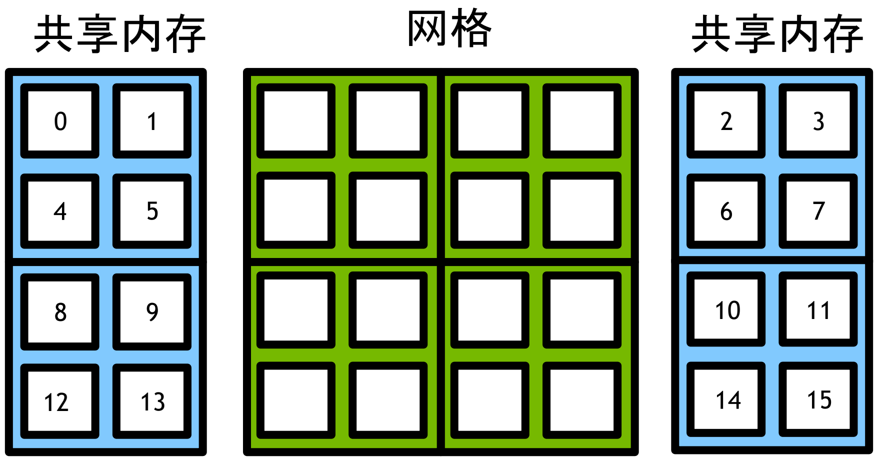




通过这种方式，我们可以实现矩阵转置，同时对全局内存进行完全合并的读取和写入。

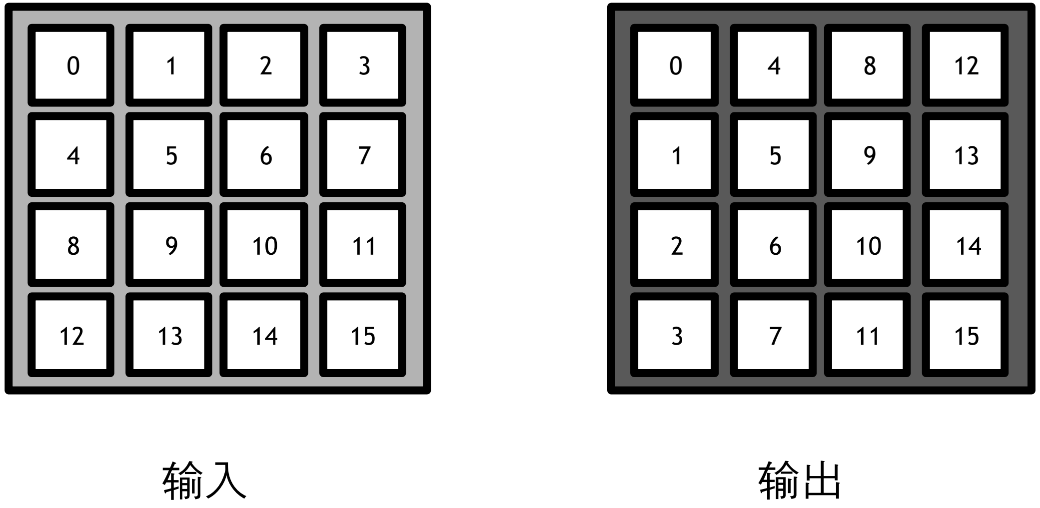
```
tile = cuda.shared.array(2,2)
x, y = cuda.grid(2)

tile[tIdx.y][tIdx.x] = in[y][x]
cuda.syncthreads()

o_x = bId.y*bDim.y + tId.x
o_y = bId.x*bDim.x + tId.y
o[o_y][o_x] = tile[tIdx.x][tIdx.y]
```





通过这种方式，我们可以实现矩阵转置，同时对全局内存进行完全合并的读取和写入。

```
tile = cuda.shared.array(2,2)
x, y = cuda.grid(2)

tile[tIdx.y][tIdx.x] = in[y][x]
cuda.syncthreads()

o_x = bId.y*bDim.y + tId.x
o_y = bId.x*bDim.x + tId.y
o[o_y][o_x] = tile[tIdx.x][tIdx.y]
```



NVIDIA

DEEP LEARNING INSTITUTE

学习更多课程，请访问 www.nvidia.cn/DLI