F

T ~~F~~

T

T ~~F~~

T ~~F~~

T

→ through put

→ SM, warp, cores

Faster accessing read-only data

~~wasting memory space~~

limited size restrits large size vectors

~~it takes 6 sec for the 10% workload cannot offload to CPU~~

~~→ 34 sec in CPU, 4 sec in GPU~~

~~⇒ $\frac{34}{4} = 8.5$ times speed up~~

$40 \times 0.9 = 36$ sec → 4sec

$40 \times 0.1 = 4$ sec (CPU)

⇓

9倍

$10 - 4 - 4 = 2$ #

$\boxed{3 \times 3 \times 3}$

$(32-2)(32-2)$

$5 \times (32-2)(32-2) \times [2 \times 27]$

a

c

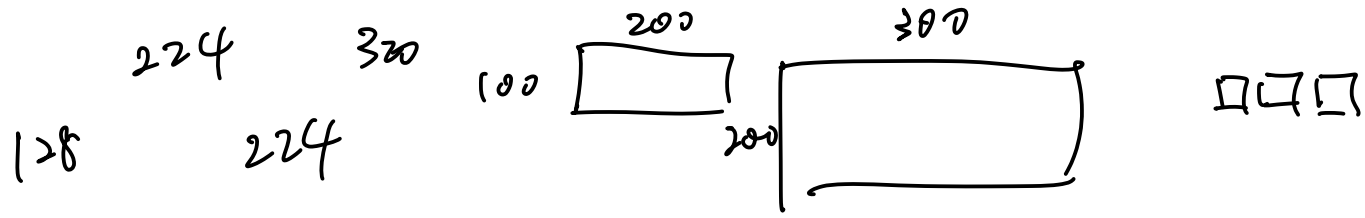
→ improving cache

b

b

threads within same wrap will access consecutive global memory

b



Ask does halo cells require FLOP ?

$(200 \times 2) \times 100 \times 300$

or $(224 \times 2) \times (128 \times 320)$?

$\frac{300}{32} \doteq 10$

$10 \times 200 \times 100 + 4 \times 200 \times 300$

$\frac{8 \times 10 \times 12}{8 \times 8 \times 8}$

$$\frac{8 \times 10 \times 12}{8 \times 8 \times 8} = 1.875$$

$$\frac{8 \times 8 \times 8 \times 1 \times 3 \times 5}{8 \times 10 \times 12} = 8$$

threadId.x 0~127, 128~255 are in same wrap.

~~1024~~ 128

global shared register register

0

int index = blockId.x · BlockElement
+ i · ELEM_Thread + threadId.x

⇒ use coalescing to load consecutive global address

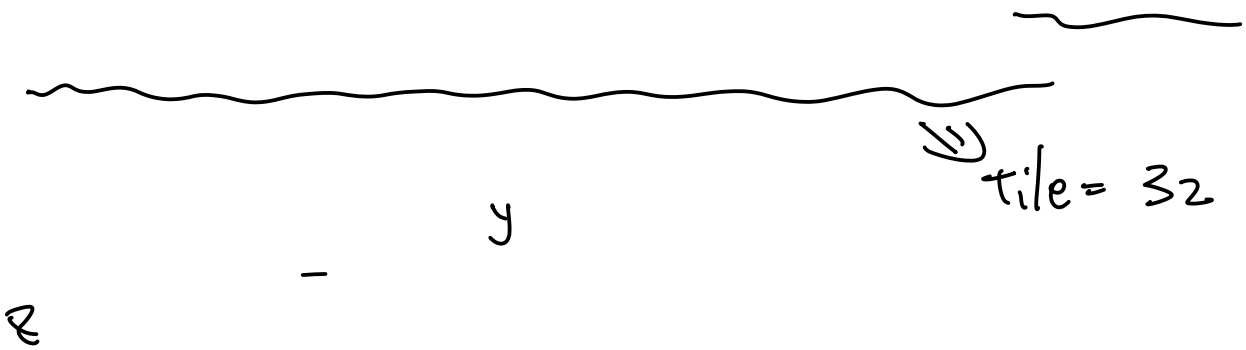
Control divergence occurs when threads within the same warp take different execution paths due to conditional branches. In such cases, the GPU serializes the execution of the divergent branches, reducing parallel efficiency.

~~32~~ 0

~~0~~ $\frac{128}{32} = 4$

0

The error arises because multiple threads update the shared variable max concurrently without proper synchronization or atomic operations, causing race conditions.



~~3~~ 32

```
__shared__ tile[TILE_WIDTH][tile_width]
int count = 9

tile[tz][ty] = in[index_z*y_size + index_y]
__syncthreads()
```

$z < index\_z + blur\_r$

$y < index\_y + blur\_y$

$z < 0$ or $z \geq row$ or $y < 0$ or $y \geq col$

if in current block

~~$0 \leq tz < tile\_width$~~

~~$0 \leq ty < tile\_width$~~

$z - b_z \times blockDim.z$ $\quad$ ~~ty~~ $y - blur\_y$

```
sum += in[z*y_size + y]

if(count > 0 && index_z < z_size && index_y < y_size)
out[index_z*y_size + index_y] = sum / count
```

```
float *in_d
float *out_d
cudaMalloc((void**) &in_d, row x col x sizeof(float))
cudaMemcpy(in_d, in, row x col x sizeof(float), HostToDevice)
dim3 dimGrid = (1, ceil(col/tileWidth), ceil(row/tileWidth))
dim3 dimBlock = (1, tileWidth, tileWidth)
blurKernel<<< dimGrid, dimBlock >>>(in_d, out_d,
cudaMemcpy(out, out_d, size, DeviceToHost) row, col, h(ur, r)
cudaFree
```

Make sure in elements are completely loaded to the tile