__shfl_down_sync(mask, val, offset)

① participating thread calling this function.

if mask对应那个线程为0 ⇒ 这个线程从别的线程处获取val, 成功吗? 或得直接返回自己?

⇓

实验来看, mask = 0. (全0)

调用线程仍然工作 返回对应线程的值.

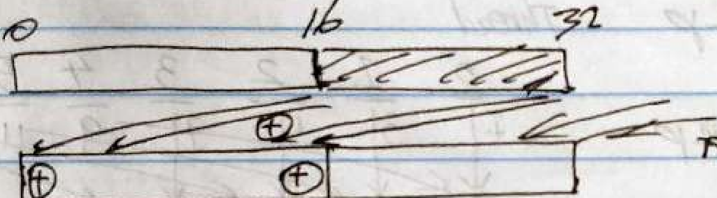
但是doc说 undefined behavior

② mask 为0xffffffff (全1), 但是有些线程不调用这个函数 (inactive)

那些线程去拿它的值后返回 是不确定的.

因为那些线程并没有提供值去shuffle

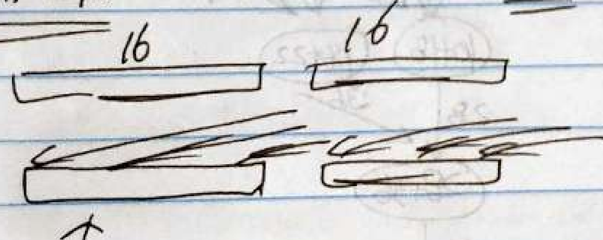
③



这些线程(16-31)要的val超出范围 直接返回自己

其它thread的

④ width = 16   16个thread为一组, 不可以理解成2个warp.



超出范围, down_sync 返回自己.

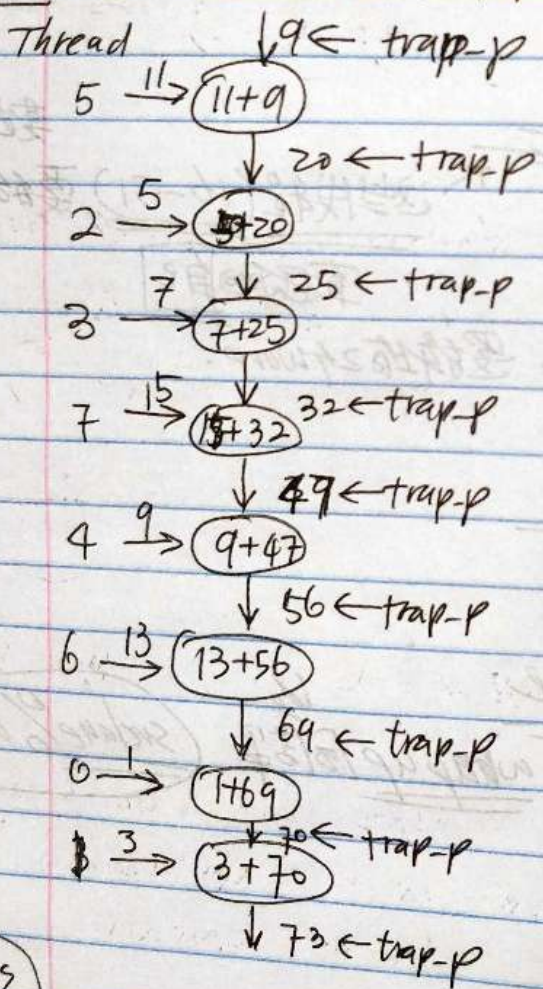
shfl_sync 返回 wrap up 后的结果 srclane % width

SIMT : Single instruction Multiple Thread (not SIMD)

because threads on an SM that are executing the same instruction may not execute simultaneously; to hide memory access latency, some threads may block while memory is accessed and other threads that have already accessed the data, may proceed with execution.

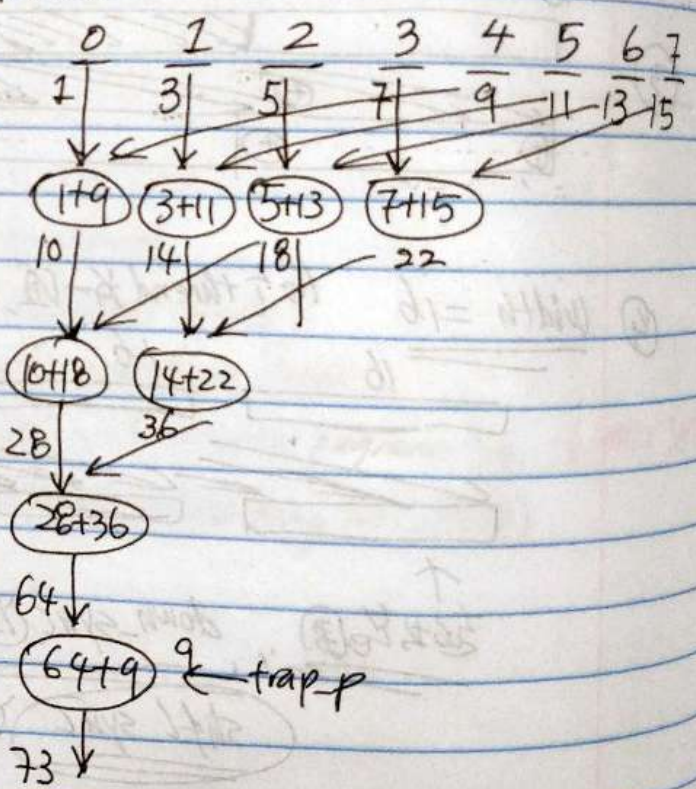
SIMD : composed of a single control unit and multiple datapaths. The control unit fetches an instruction from memory and broadcast it to the datapaths. Each data path either executes the instruction on its data or is idle.

Basic Sum: old value ← AtomicAdd(trap_p, val);



t threads
t value
→ t sequential additions

Tree-structured Sum



$\lceil \log_2 t \rceil + 1$

warpSize

lane = threadIdx.x % warpSize;

the warp shuffle function allows threads in a warp to read from registers used by another thread in the same warp.

```
__device__ float __shfl_down_sync(
        unsigned mask,
        float var,
        unsigned diff,          // l+diff (diff >= 0).
        int width = warpSize );
```

(mask →) A bit representing the thread's lane must be set for each participating thread to ensure that all of the threads in the call have converged — ie arrived at the call — before any thread begins executing the call to __shfl_down_sync.

thread l ⟵ value ⟵ thread l+diff

① what happens if thread l calls __shfl_down_sync, but l+diff doesn't??
↳ the value returned by the call on thread l is **undefined**
(l+diff thread is inactive due to if branch) ← 返回0 好像.

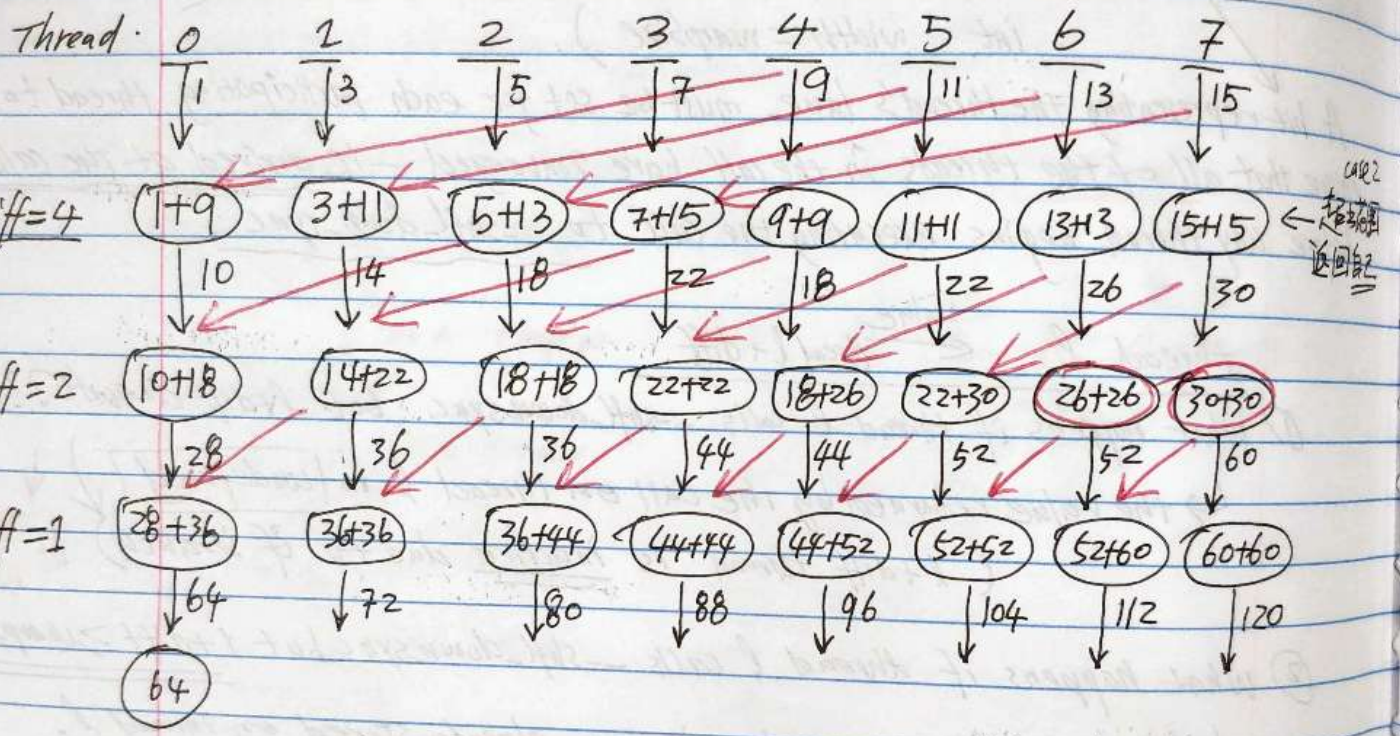
② what happens if thread l calls __shfl_down_sync but l+diff ≥ warpsize
↳ the call returns the value in var already stored on thread l.
(original value of var) 用width 来控制

③ what happens if thread l calls __shfl_down_sync but l+diff < warpsize and l+diff > largest lane in the warp. (due to thread block size is not multiplier of warp size). the last warp has no 32 threads (<32).
similar to ①, some threads are inactive.
so value returned by the call is also (undefined) 返回0 好像

被 (mask 指定的 participating threads 返回 ---- 像上面描述)

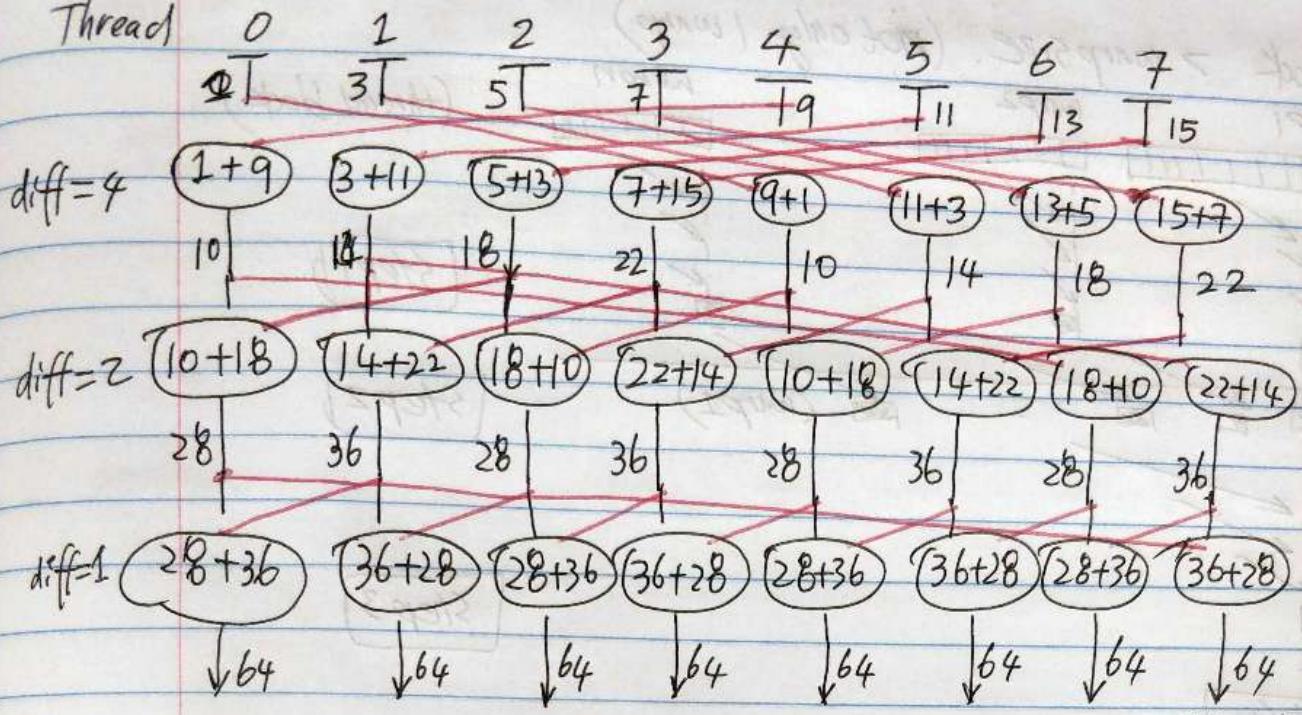
未被 mask 指定的 participating threads 返回 0 (仍然为active threads, 但 mask 指定为 0).

```
__device__ float warpSum(float val) {
    unsigned mask = 0xffffffff;
    for(int diff = warpSize/2; diff > 0; diff >>= 1) {
        val += __shfl_down_sync(mask, val, diff);
    }
    return val;
}
```




```
int my_i = blockDim.x * blockIdx.x + threadIdx.x;
float my_trap = 0.0f;
if (0 < my_i && my_i < n) {
    float my_x = a + my_i * h;
    my_trap = f(my_x);
}
float result = Warp_Sum(my_trap);
if (threadIdx.x == 0) atomicAdd(trap_p, result);
```

(one thread block only a warp)



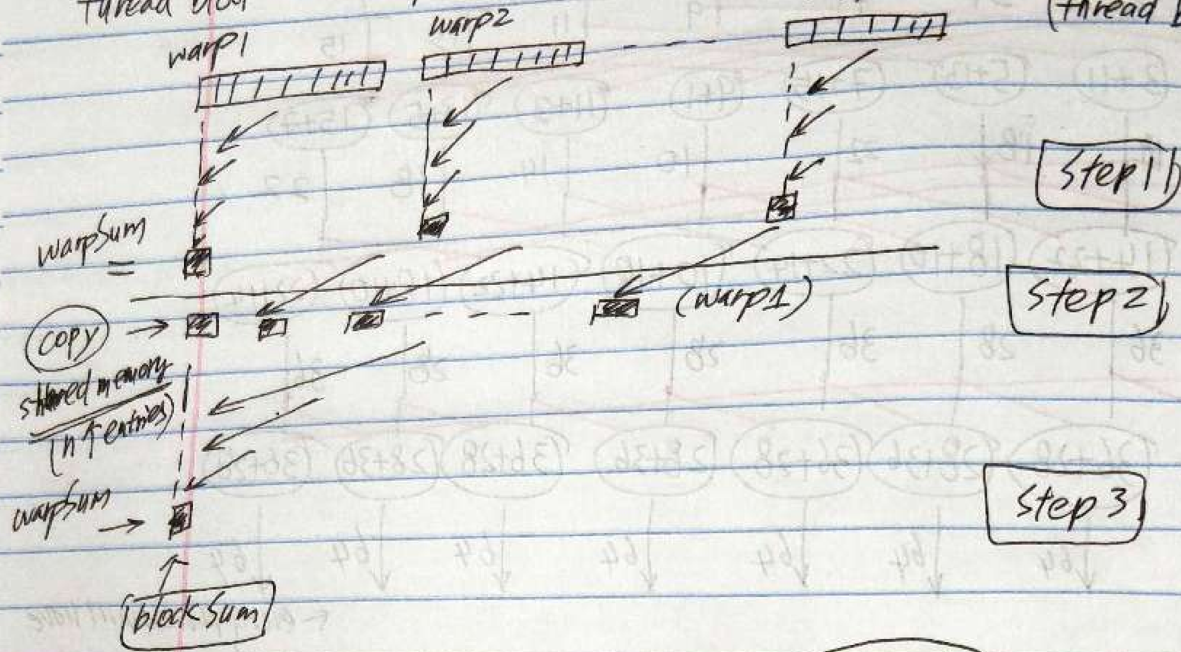
← every thread will have final sum.

```
__device__ float shared_mem_sum(float shared_vals[]) {
    int my_lane = threadIdx.x % warpSize;
    for(int diff=warpSize/2; diff>0; diff>>=1) {
        int source = (my_lane+diff) % warpSize;
        shared_vals[my_lane] += shared_vals[source];
    }
    return shared_vals[my_lane];
}
```

either global memory or shared memory

//dissemination sum using shared memory

thread block > warpsize. (not only 1 warp)



(pytorch code)

```
T BlockReduceSum(T val, T* shared) {
    const int laneid = threadIdx.x % warpsize;
    const int warpid = threadIdx.x / warpsize;
    val = WarpReduceSum(val);
    __syncthreads();   // 好像没有要!!!
    if (laneid == 0)
        shared[warpid] = val;
    __syncthreads();
    val = (threadIdx.x < blockDim.x / warpSize) ? shared[laneid] : T(0);
    if (warpid == 0)
        val = warpReduceSum(val);
    return val;
}
```

blockDim.x / warpSize → how many warps in thread block?

(warp)

```
int sum = 0;
for (int i = blockIdx.x * blockDim.x + threadIdx.x;
     i < N;
     i += blockDim.x * gridDim.x) {
    sum += in[i];
}
sum = blockReduceSum(sum);
if (threadIdx.x == 0)
    atomicAdd(out, sum);
```

benchmark

Also good.

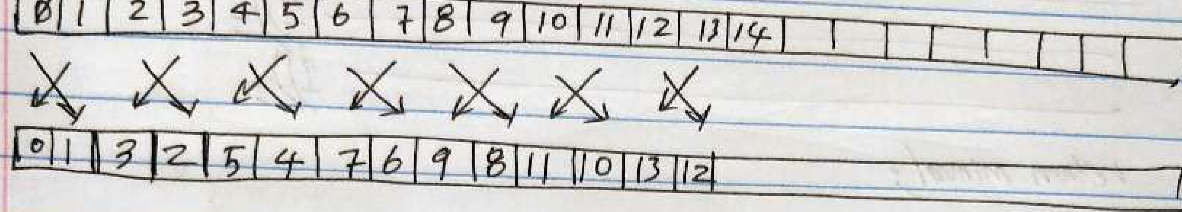
skip block-level processing

```
int sum = 0;
for (int i = blockIdx.x * blockDim.x + threadIdx.x;
     i < N;
     i += blockDim.x * gridDim.x) {
    sum += in[i];
}
sum = warpReduceSum(sum);
if (threadIdx.x & (WARP_SIZE - 1) == 0)
    atomicAdd(out, sum);
```

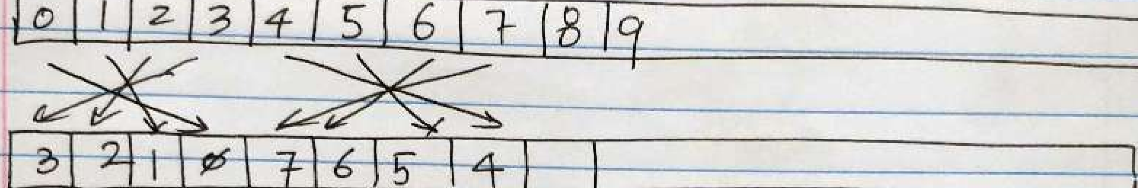
每32个threads 每warp, 第1个thread

__shfl_xor_sync (unsigned mask, T var, int laneMask, int width=WarpSize)

laneMask=1

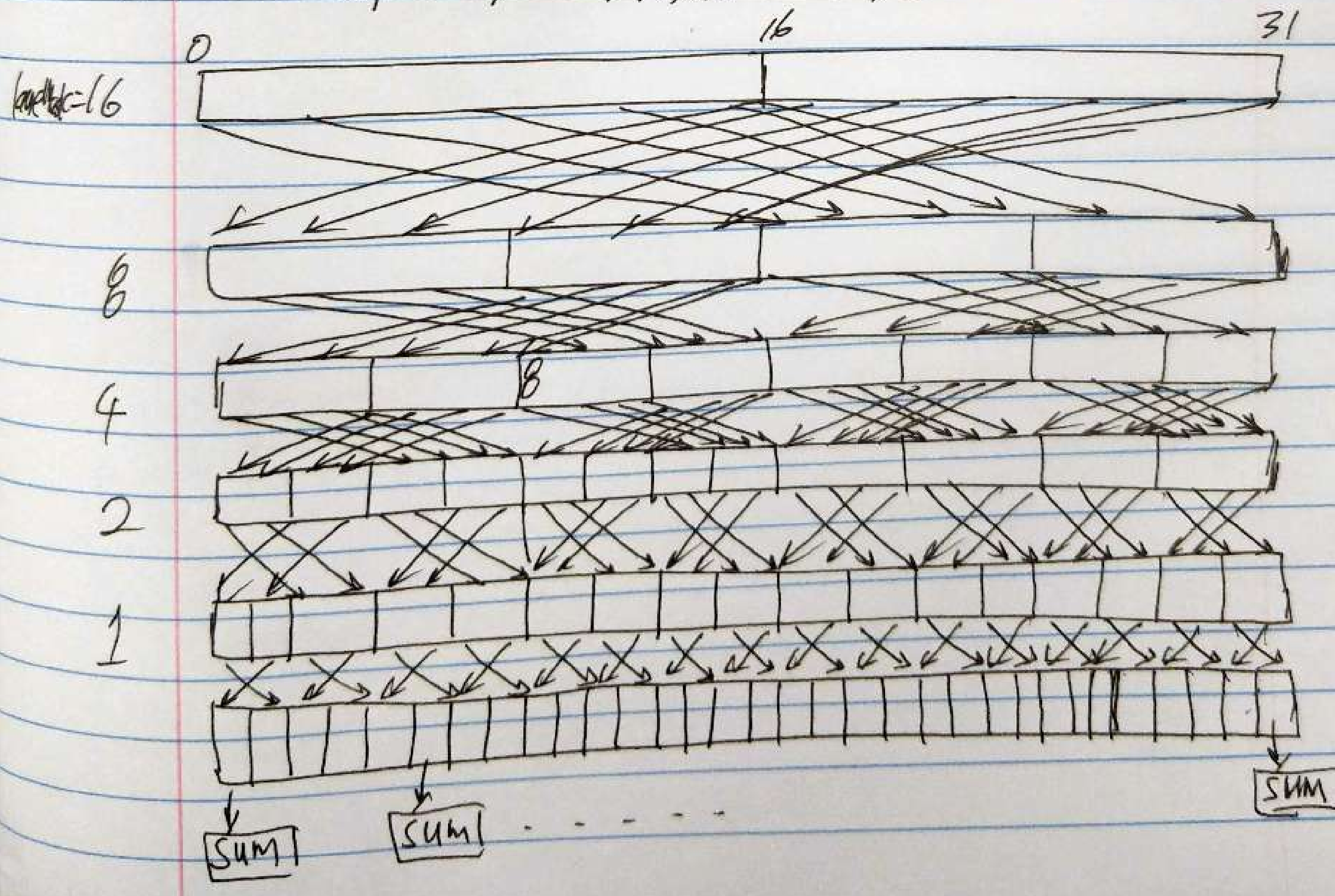


laneMask=3




```
sum += __shfl_xor_sync(0xffffffff, sum, 16);
sum += __shfl_xor_sync(0xffffffff, sum, 8);
sum += __shfl_xor_sync(0xffffffff, sum, 4);
sum += __shfl_xor_sync(0xffffffff, sum, 2);
sum += __shfl_xor_sync(0xffffffff, sum, 1);
```

也可用来求得 min/product、
所有线程最后都有 value/meaningful



```
__device__ __forceinline__ int warpReduceMin(int minVal)
    minval = min(minval, __shfl_xor_sync(0xffffffff, minval, 16));
    minval = min(minval, __shfl_xor_sync(______, ______, 8));
    ____________________________________________________ 4));
    ____________________________________________________ 2));
    ____________________________________________________ 1));

    return minval;
}
```

min ⇒ val *= __shfl_xor_sync ---- 16→8→4→2→1.