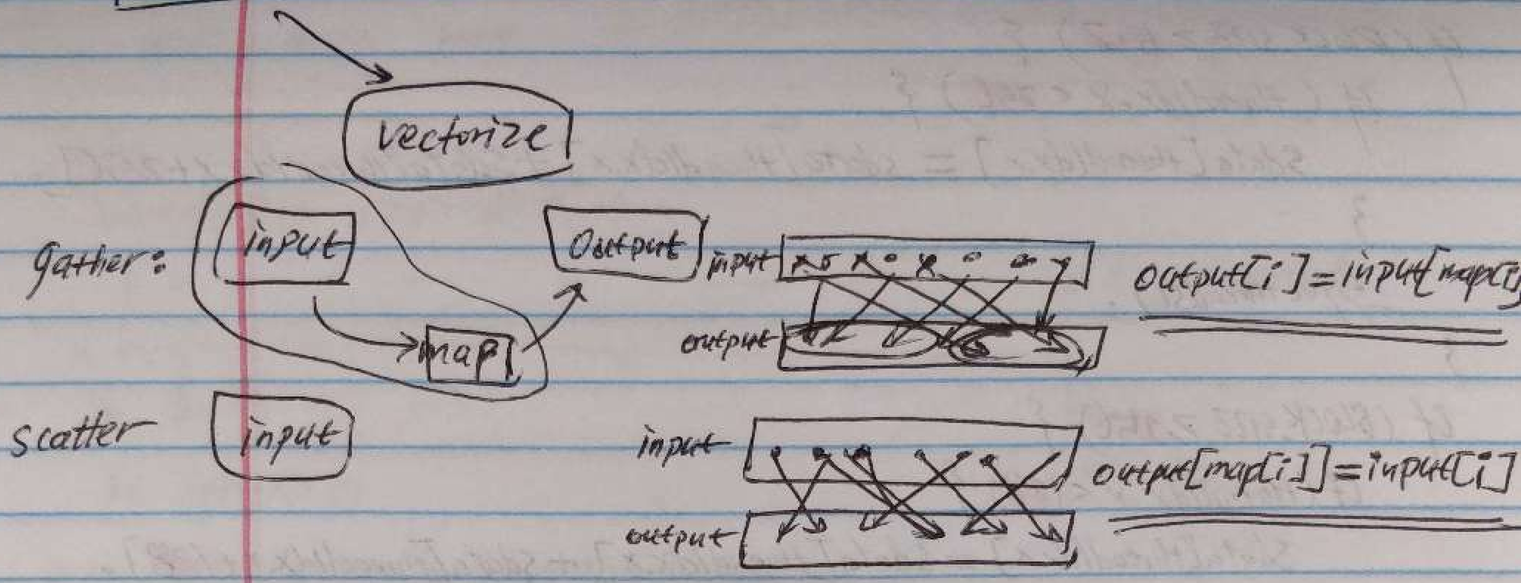
thrust

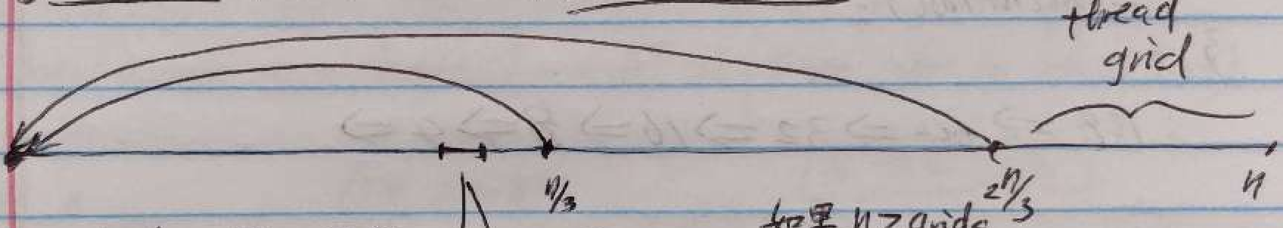
gather/scatter　　scan　　reduce



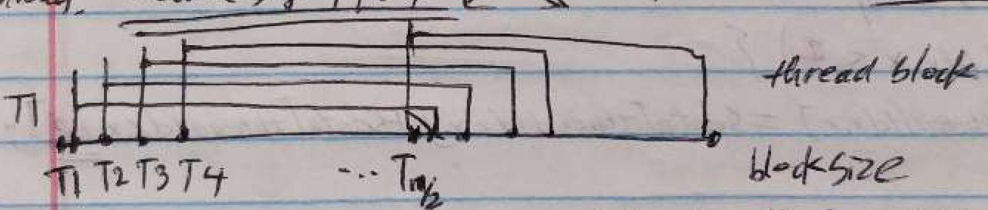
output[i] = input[map[i]]

output[map[i]] = input[i]

reduce 类似于 accumulate 但不保证 order. accumulate 保证



每个block thread. reduce出一个结果　　如果 n > grids



① 一半的Threads, 每个thread reduce $(i, i+m/2)$ 2个值

⇒ sync block

② 问题域缩小到 $m/2$.

再一半的threads. 每个threads reduce

⇒ sync block

↳ 直到只剩下一个 thread reduce 出最终结果.

⇒ 每reduce一次只有一半的线程干活。

```
if (BLOCK_SIZE >= 512) {
    if (threadIdx.x < 256) {
        sdata[threadIdx.x] = sdata[threadIdx.x] + sdata[threadIdx.x+256];
    }
    __syncthreads();
}
if (BLOCK_SIZE >= 256) {
    if (threadIdx.x < 128) {
        sdata[threadIdx.x] = sdata[threadIdx.x] + sdata[threadIdx.x+128];
    }
    __syncthreads();
}
  : 128 => 64 => 32 => 16 => 8 => 4 =>

if (BLOCK_SIZE >= 2) {
    if (threadIdx.x < 1) {
        sdata[threadIdx.x] = sdata[threadIdx.x] + sdata[threadIdx.x+1];
    }
    __syncthreads();
}
if (threadIdx.x == 0)
    block_results[blockIdx.x] = sdata[threadIdx.x];
```

//最后将reduce结果写入全局memory
每个block对应一个reduce结果.

在外围kernel返回时.
在host层面reduce最后结果。

inclusive_scan

1 2 3 4 5

1 3 6 10 15

exclusive_scan

1 2 3 4 5

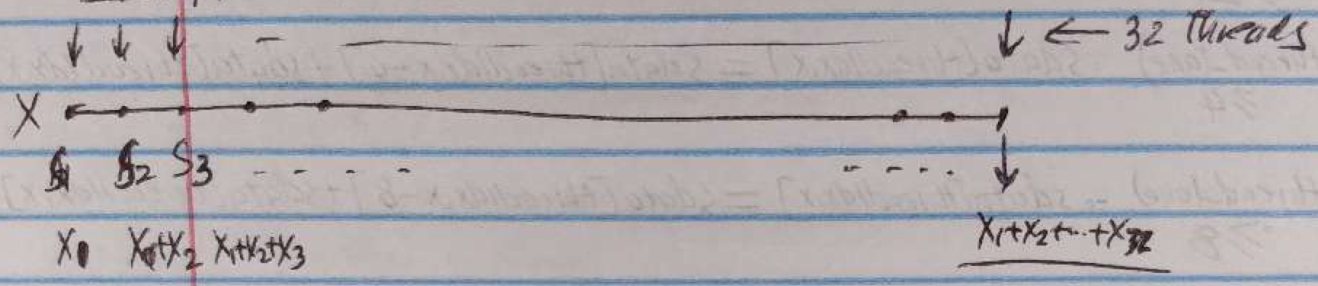
∅ 1 3 6 10

warp size = 32

8x warps 1block

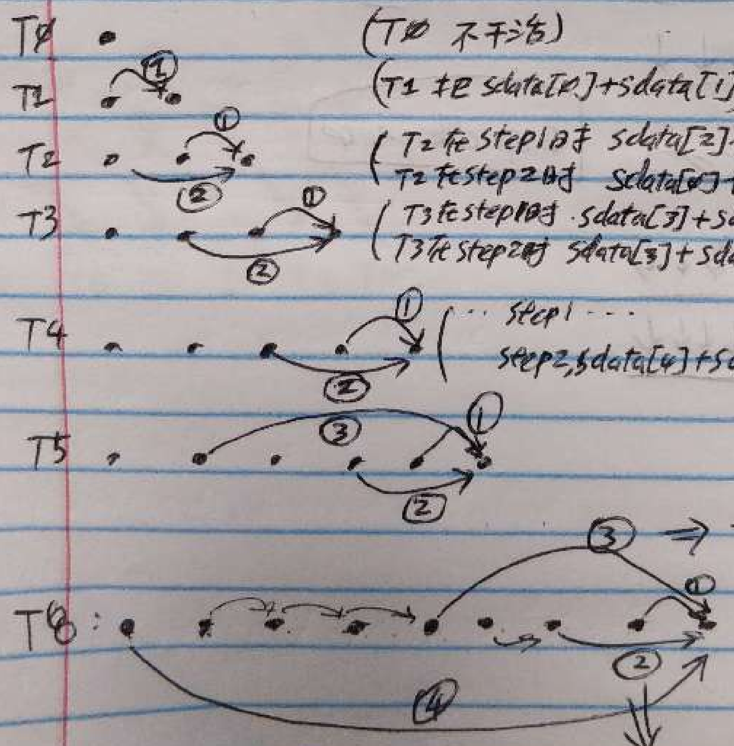
warp1
warp2
warp3
warp8

从warp为单位，拆分问题域 n



- 每个warp计算出一个 carry = 最后一个加和结果 $\sum_{i=1}^{32} x_i$.
- 如何计算32个数的数组的 inclusive_scan. sdata ⇐ block shared memory
  - sdata[threadIdx.x] = input[i]; ← 每个线程copy input to sdata (local)

R → C → W



(T0 不干活)

(T1 把 sdata[0]+sdata[1]) → Read 在T1的W之前

(T2在step1时 sdata[2]+sdata[1]
T2在step2时 sdata[0]+sdata[2])

(T3在step1时 sdata[3]+sdata[2] → Read 在T2的W之前
T3在step2时 sdata[3]+sdata[1]) → step2时 T1的step1已结束，故 sdata[1]已更新为 sdata[0]+sdata[1]

(... step1 ...
step2, sdata[4]+sdata[2]) → sdata[2]已被T2已更新. 故不需要再加 sdata[0] sdata[1] ~~sdata[2]~~.

③ ⇒ T4已在step 2计算完.

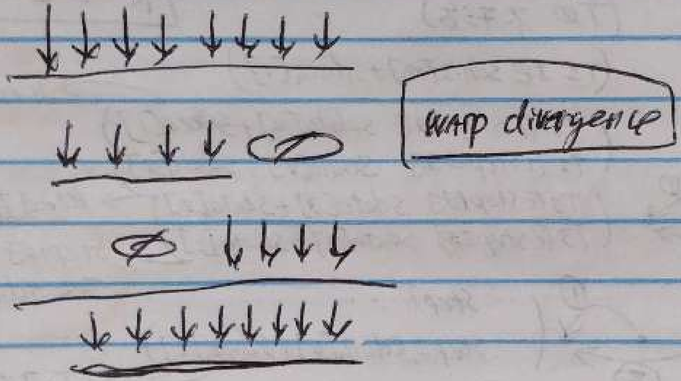
④ ⇒ T6已在step1计算完.

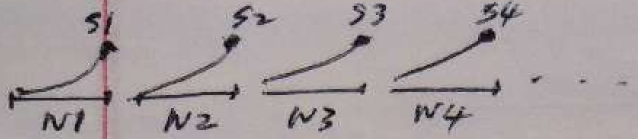
代码 ⇒

thread_lane, volatile T* sdata.

```
if (thread_lane ≥1)  sdata[threadIdx.x] = sdata[threadIdx.x-1]+sdata[threadIdx.x];
if (thread_lane ≥2)  sdata[threadIdx.x] = sdata[threadIdx.x-2]+sdata[threadIdx.x];
if (thread_lane ≥4)  sdata[threadIdx.x] = sdata[threadIdx.x-4]+sdata[threadIdx.x];
if (thread_lane ≥8)  sdata[threadIdx.x] = sdata[threadIdx.x-8]+sdata[threadIdx.x];
if (thread_lane ≥16) sdata[threadIdx.x] = sdata[threadIdx.x-16]+sdata[threadIdx.x];
```

当thread_lane：当前thread在32条threads(warp)中的第n条 ≤ (warpsize-1)
=3时，前2条if也会执行，其它的不会执行，但是会跟其thread同步到达最后。





- 当每个warp计算好最后累加和之后就放carry.

除了第1个warp之外

- 对S1, S2, S3, S4 ⇒ 也计算 inclusive_scan

  因为 W2 需要加S1 ⇒ S1+S2

  W3 需要加上 S1+S2 ⇒ ~~S1+~~. inplace_scan. carry.

  W4 需要加上 S1+S2+S3.

- 从W2开始每个元素都要加 S1.

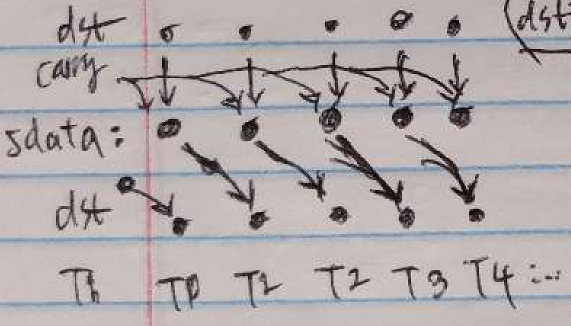
  W3开始每个元素都要加 S1+S2.

  W4开始每个元素都要加 S1+S2+S3.

```
                              ←只会loop一次
for( i=begin;  i<end;  i+=32) {
    sdata[threadIdx.x] = carry + dst[i];
    dst[i] = (thread_lane==0) ? first[warp_lane] : sdata[threadIdx.x - 1];
}
```

前个线程已帮忙计算了.

sdata: block thread shared memory

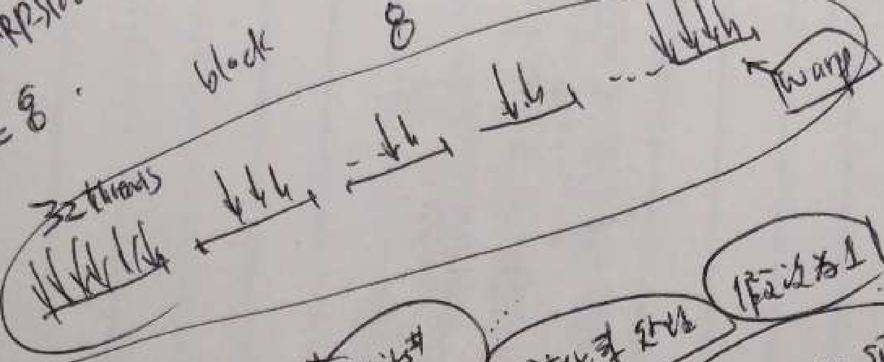


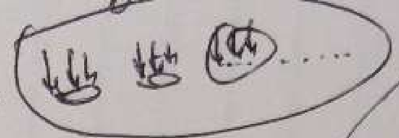
dst[i] 已累加过

WARPS_PER_BLOCK = BLOCKSIZE/WARPSIZE

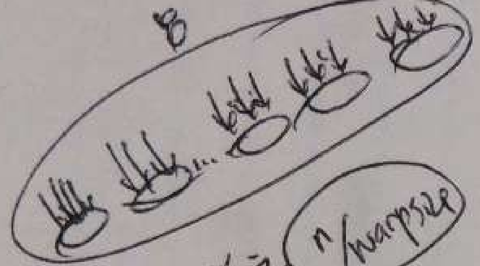
= 256/32 = 8.

block 8





8



一个warp作为一个块的事之 n/warpsize

num_iters = ~~warp~~ units# / warps#

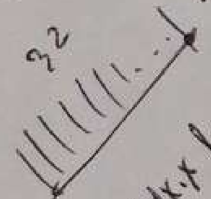
分配任务 线程

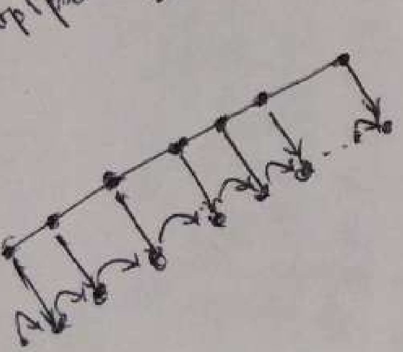
假设为1

per-warp size

iteral_size = WARP_SIZE*num_iters;

×8 ...





threadlane

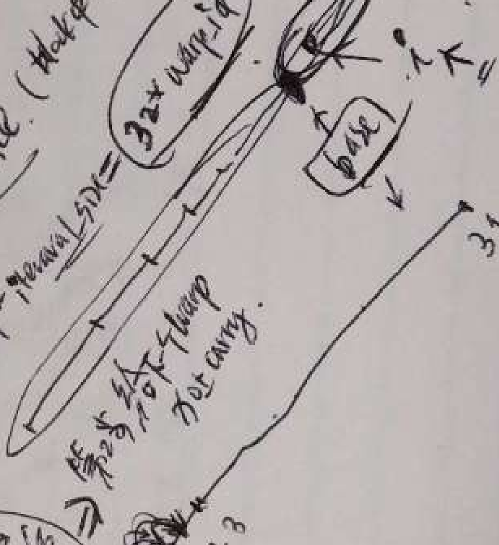
warp中第几个thread = threadIdx.x & warp_size (%32).

warp_id = global thread id / warp_size. (全局第几个warp)

warp_lane = threadIdx.x / warp_size. (block中第几个warp) local

b04 = warp_id * iteral_size = 32×warp_id

i ← = global thread index.



第2号给下一个warp 加上carry.

carry → 每一个warp一个carry

一个block 有几个warp — BLOCKSIZE/warpsize.

warp中最后的线程 threadlane=0 才更新

分治法

树形 两边各自计算.

同边每位置都加上

左边最后一个元素 carry

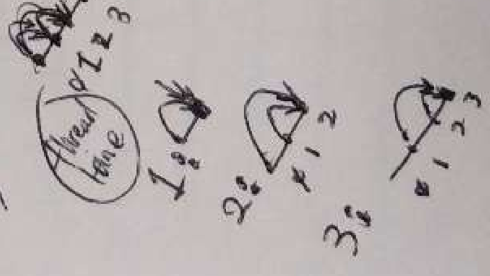
每个warp各自计算.

然后每个warp的最后

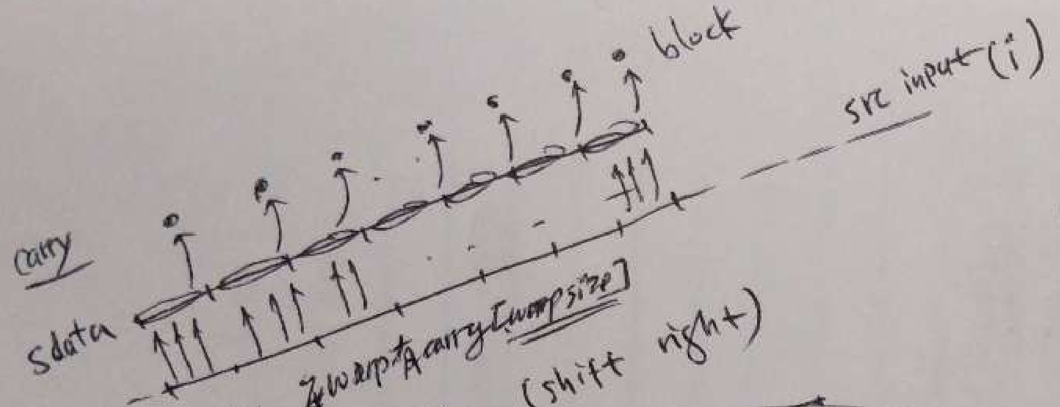
计算得到carry 传递

⇒

threadlane 0 1 2 3



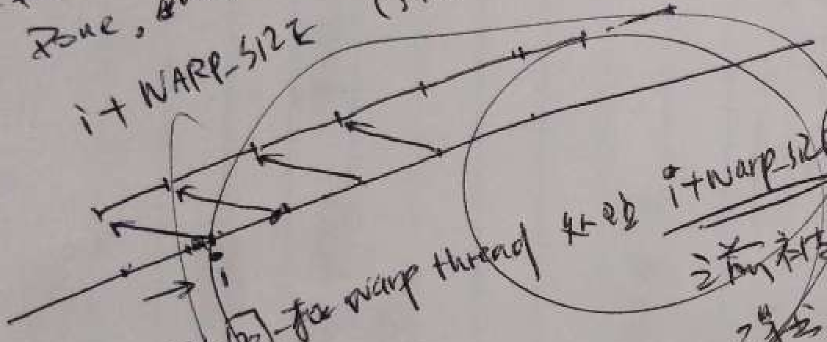
warp中最后的线程 data 加上 carry[warp_lane]



Done，每warp有个carry[warp size] (shift right)

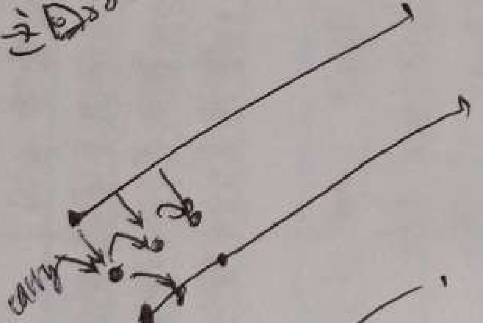
①

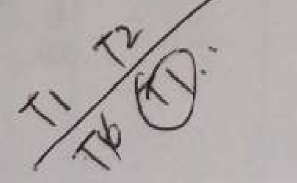
② i+WARP_SIZE

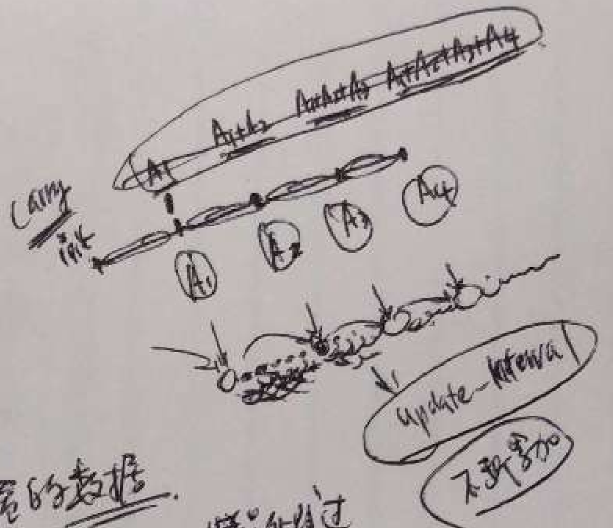


下一代另(同)一个 warp thread 处理 i+warp_size 位置的数据，之前存储 ~~warp_index~~ -1的偏移处理过 得到 carry.

这一iteration 到位 但是需要加上 carry 上一范围的 carry 这里没问题也是对的：但

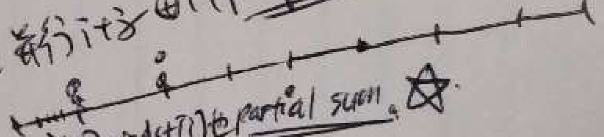






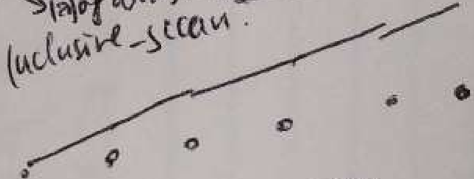
Sum of each warp

① 先并行计算每个 warp size .carry



→ 得到dst[i]也partial scan. ☆.

② inclusive_scan.



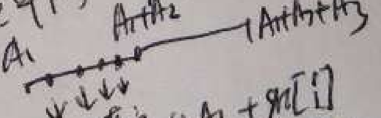
再并行 inclusive_scan.

A1 A2 --- An

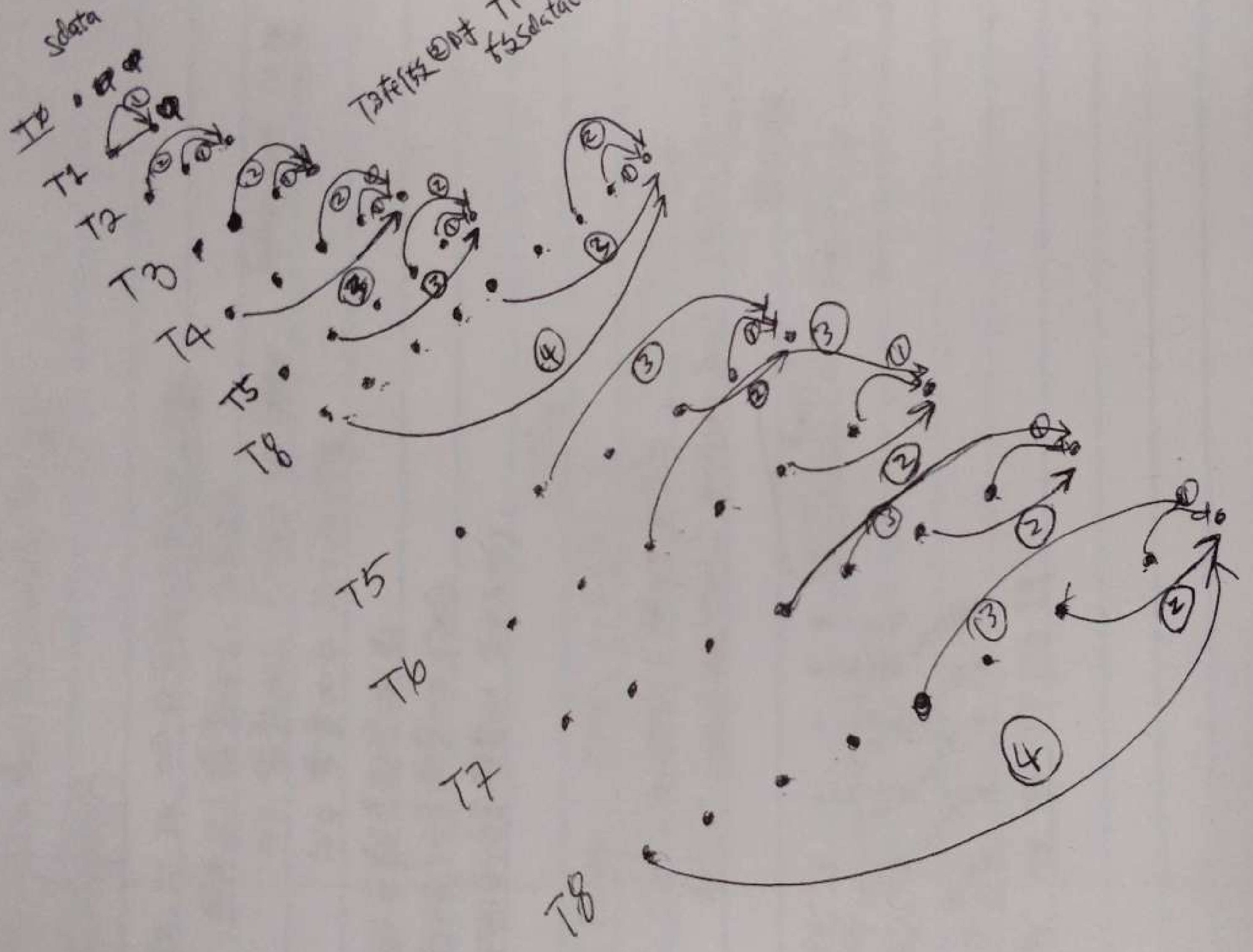
A1 A1+A2 A1+A2+A3 --- A1+A2+...+An

⇒ A1 A1+A2

③ 处理中间warp interval



处理 从A1 + [illegible][i]



T3在做②时 T1已完成①，于是sdata[1]已经是 sdata[0]+sdata[1] 之值