

thrust

gather/scatter

scan

reduce

vectorize

gather:

input

output

map



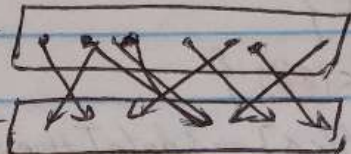
$output[i] = input[map[i]]$

scatter

input

input

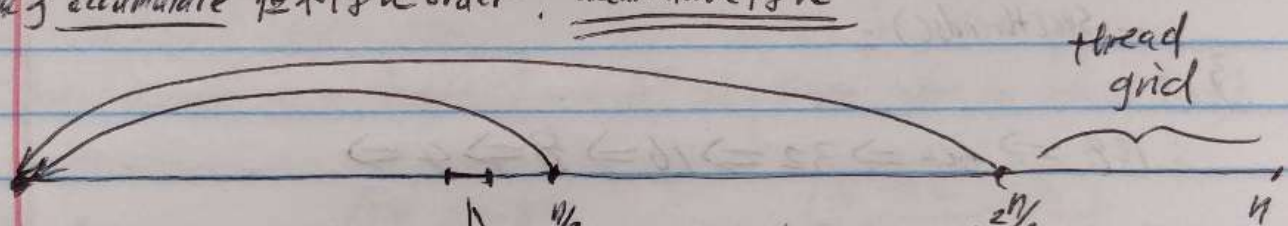
output



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

reduce

类似于 accumulate 但不保证 order, accumulate 保证



每个 block 1 thread

reduce 共享一个结果

zoom in

如果 $n > grids$



thread block

block size

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

\Rightarrow sync block

② 问题域缩小到 $m/2$

再一半的 threads, 每个 threads reduce

\Rightarrow sync block

③ 直到剩下 4 threads reduce 为最终结果

\Rightarrow 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();
}

```

$128 \Rightarrow 64 \Rightarrow 32 \Rightarrow 16 \Rightarrow 8 \Rightarrow 4 \Rightarrow$

```

if (BLOCK_SIZE > 2) {
    if (threadidx.x < 1) {
        sdata[threadidx.x] = sdata[threadidx.x] + sdata[threadidx.x + 1];
    }
    --syncthreads();
}

```

```

if (threadidx.x == 0)
    block_result[blockidx.x] = sdata[threadidx.x];

```

//最后将reduce结果存入全局memory
每个block执行一个reduce结果。

在外围 kernel 返回时。

在 host 后面 reduce 最后结果。

WARD SIZE = 32

1 2 3 4 5

↓ ↓ ↓ ↓ ↓
2 3 6 10 15

8x waps 1 block

Warp1 Warp2 Warp3

$\emptyset \rightarrow 1 \rightarrow 3 \rightarrow 6 \rightarrow 10$

war 3

以 map 为单位, 拆分问题域 n

Diagram illustrating the reduction of 32 threads to a single thread. A horizontal line represents a sequence of operations. On the left, three downward arrows point to the line, with 'X' written below them. Below the line, the sequence 'S1 S2 S3 ...' is written. On the right, a single downward arrow points to the line, with '32 threads' written next to it. Below the line, the sequence 'X0 X1 X2 X3 ... X31 X32' is written, with 'X1+X2+...+X32' written below it.

• 每个warp产生一个 carry = 最后一个加和结果 $\sum_{i=1}^{32} x_i$

• 如何计算这32个数组的 inclusive_scan. $sdata \leftarrow \text{block shared memory}$

- $Sdata[threadIdx.x] = input[i]$, ← 每个线程 copy input to sdata (local)

Diagram illustrating a race condition between two threads, T1 and T2, accessing shared memory.

Thread T1:

- Step 1: Read $data[0]$ and $data[1]$.
- Step 2: Calculate $sdata[0] + sdata[1]$.
- Step 3: Update $data[0]$ and $data[1]$ with the result.

Thread T2:

- Step 1: Read $data[2]$ and $data[3]$.
- Step 2: Calculate $sdata[2] + sdata[3]$.
- Step 3: Update $data[0]$ and $data[1]$ with the result.

The diagram shows that T2's update occurs before T1's, leading to a race condition. The final state of the memory is shown as $data[0] = 10$ and $data[1] = 20$, which is the result of T2's update, not T1's.

Diagram illustrating a sequence of steps (1, 2, 3, 4) and their corresponding states (T0, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15, T16, T17, T18, T19, T20, T21, T22, T23, T24, T25, T26, T27, T28, T29, T30, T31, T32, T33, T34, T35, T36, T37, T38, T39, T40, T41, T42, T43, T44, T45, T46, T47, T48, T49, T50, T51, T52, T53, T54, T55, T56, T57, T58, T59, T60, T61, T62, T63, T64, T65, T66, T67, T68, T69, T70, T71, T72, T73, T74, T75, T76, T77, T78, T79, T80, T81, T82, T83, T84, T85, T86, T87, T88, T89, T90, T91, T92, T93, T94, T95, T96, T97, T98, T99, T100, T101, T102, T103, T104, T105, T106, T107, T108, T109, T110, T111, T112, T113, T114, T115, T116, T117, T118, T119, T120, T121, T122, T123, T124, T125, T126, T127, T128, T129, T130, T131, T132, T133, T134, T135, T136, T137, T138, T139, T140, T141, T142, T143, T144, T145, T146, T147, T148, T149, T150, T151, T152, T153, T154, T155, T156, T157, T158, T159, T160, T161, T162, T163, T164, T165, T166, T167, T168, T169, T170, T171, T172, T173, T174, T175, T176, T177, T178, T179, T180, T181, T182, T183, T184, T185, T186, T187, T188, T189, T190, T191, T192, T193, T194, T195, T196, T197, T198, T199, T200, T201, T202, T203, T204, T205, T206, T207, T208, T209, T210, T211, T212, T213, T214, T215, T216, T217, T218, T219, T220, T221, T222, T223, T224, T225, T226, T227, T228, T229, T230, T231, T232, T233, T234, T235, T236, T237, T238, T239, T240, T241, T242, T243, T244, T245, T246, T247, T248, T249, T250, T251, T252, T253, T254, T255, T256, T257, T258, T259, T260, T261, T262, T263, T264, T265, T266, T267, T268, T269, T270, T271, T272, T273, T274, T275, T276, T277, T278, T279, T280, T281, T282, T283, T284, T285, T286, T287, T288, T289, T290, T291, T292, T293, T294, T295, T296, T297, T298, T299, T300, T301, T302, T303, T304, T305, T306, T307, T308, T309, T310, T311, T312, T313, T314, T315, T316, T317, T318, T319, T320, T321, T322, T323, T324, T325, T326, T327, T328, T329, T330, T331, T332, T333, T334, T335, T336, T337, T338, T339, T340, T341, T342, T343, T344, T345, T346, T347, T348, T349, T350, T351, T352, T353, T354, T355, T356, T357, T358, T359, T360, T361, T362, T363, T364, T365, T366, T367, T368, T369, T370, T371, T372, T373, T374, T375, T376, T377, T378, T379, T380, T381, T382, T383, T384, T385, T386, T387, T388, T389, T390, T391, T392, T393, T394, T395, T396, T397, T398, T399, T400, T401, T402, T403, T404, T405, T406, T407, T408, T409, T410, T411, T412, T413, T414, T415, T416, T417, T418, T419, T420, T421, T422, T423, T424, T425, T426, T427, T428, T429, T430, T431, T432, T433, T434, T435, T436, T437, T438, T439, T440, T441, T442, T443, T444, T445, T446, T447, T448, T449, T450, T451, T452, T453, T454, T455, T456, T457, T458, T459, T460, T461, T462, T463, T464, T465, T466, T467, T468, T469, T470, T471, T472, T473, T474, T475, T476, T477, T478, T479, T480, T481, T482, T483, T484, T485, T486, T487, T488, T489, T490, T491, T492, T493, T494, T495, T496, T497, T498, T499, T500, T501, T502, T503, T504, T505, T506, T507, T508, T509, T510, T511, T512, T513, T514, T515, T516, T517, T518, T519, T520, T521, T522, T523, T524, T525, T526, T527, T528, T529, T530, T531, T532, T533, T534, T535, T536, T537, T538, T539, T540, T541, T542, T543, T544, T545, T546, T547, T548, T549, T550, T551, T552, T553, T554, T555, T556, T557, T558, T559, T560, T561, T562, T563, T564, T565, T566, T567, T568, T569, T570, T571, T572, T573, T574, T575, T576, T577, T578, T579, T580, T581, T582, T583, T584, T585, T586, T587, T588, T589, T590, T591, T592, T593, T594, T595, T596, T597, T598, T599, T600, T601, T602, T603, T604, T605, T606, T607, T608, T609, T610, T611, T612, T613, T614, T615, T616, T617, T618, T619, T620, T621, T622, T623, T624, T625, T626, T627, T628, T629, T630, T631, T632, T633, T634, T635, T636, T637, T638, T639, T640, T641, T642, T643, T644, T645, T646, T647, T648, T649, T650, T651, T652, T653, T654, T655, T656, T657, T658, T659, T660, T661, T662, T663, T664, T665, T666, T667, T668, T669, T670, T671, T672, T673, T674, T675, T676, T677, T678, T679, T680, T681, T682, T683, T684, T685, T686, T687, T688, T689, T690, T691, T692, T693, T694, T695, T696, T697, T698, T699, T700, T701, T702, T703, T704, T705, T706, T707, T708, T709, T710, T711, T712, T713, T714, T715, T716, T717, T718, T719, T720, T721, T722, T723, T724, T725, T726, T727, T728, T729, T730, T731, T732, T733, T734, T735, T736, T737, T738, T739, T740, T741, T742, T743, T744, T745, T746, T747, T748, T749, T750, T751, T752, T753, T754, T755, T756, T757, T758, T759, T760, T761, T762, T763, T764, T765, T766, T767, T768, T769, T770, T771, T772, T773, T774, T775, T776, T777, T778, T779, T780, T781, T782, T783, T784, T785, T786, T787, T788, T789, T790, T791, T792, T793, T794, T795, T796, T797, T798, T799, T800, T801, T802, T803, T804, T805, T806, T807, T808, T809, T810, T811, T812, T813, T814, T815, T816, T817, T818, T819, T820, T821, T822, T823, T824, T825, T826, T827, T828, T829, T830, T831, T832, T833, T834, T835, T

(代码) \Rightarrow

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) 中的第几号 $\< (\text{warp_size}-1)$
= 3 时, 前 2 条 thread 也会执行, 其后的不会执行, 但是会跟其 thread 同步到达最后。

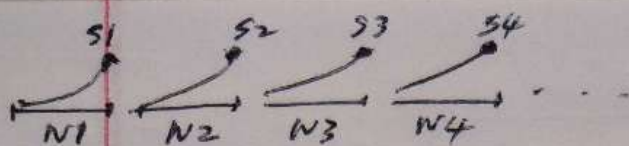
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

↓ ↓ ↓ ↓ ○

warp divergence

○ ↓ ↓ ↓ ↓

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓



- 当每个warp迭代时，累加和会翻倍，故 carry

(除第1个warp外)

- 对 $S1, S2, S3, S4 \Rightarrow$ 也是 inclusive-scan

因为 $W2$ 需要加上 $S1 \Rightarrow S1+S2$

$W3$ 需要加上 $S1+S2 \Rightarrow$ ~~$S1+S2$~~

Inplace-scan, carry

$W4$ 需要加上 $S1+S2+S3$

- 从 $W2$ 开始每个线程都要加 $S1$.

$W3$ 开始每个线程都要加 $S1+S2$.

$W4$ 开始每个线程都要加 $S1+S2+S3$.

\leftarrow 0.5 loop-1x

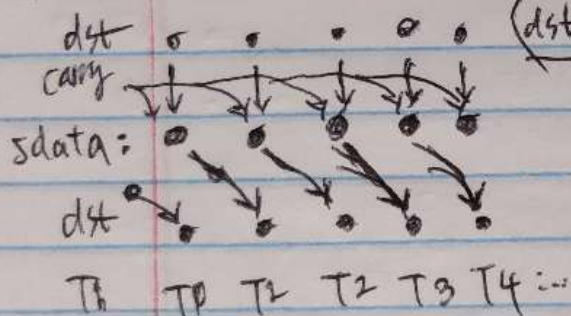
for (i = begin; i < end; i += 32) {

$sdata[threadIdx.x] = carry + dst[i];$

$dst[i] = (threadIdx.x == 0) ? first[warpLane] : sdata[threadIdx.x - 1];$

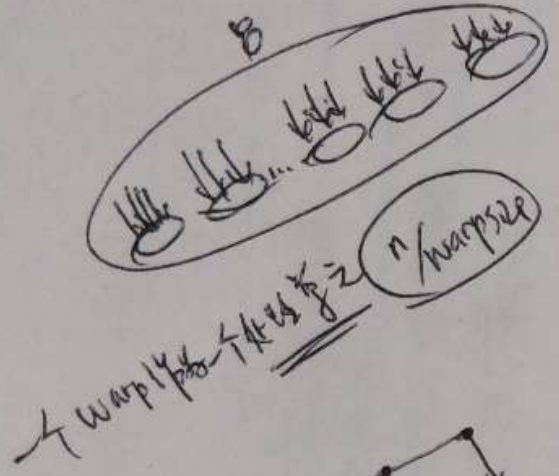
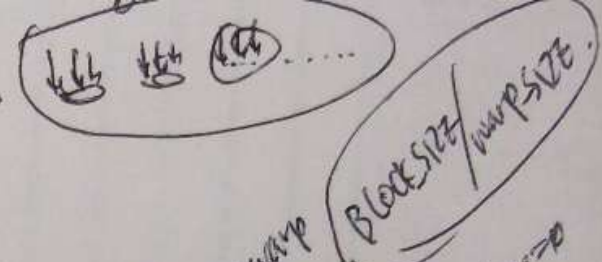
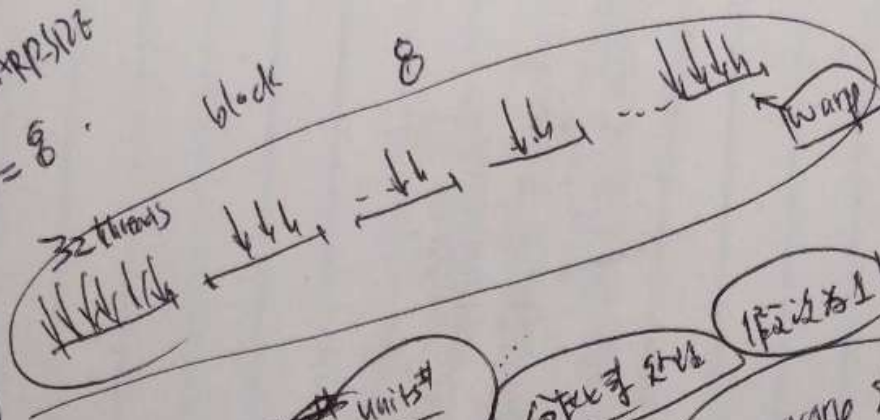
}

每个线程都要加4位



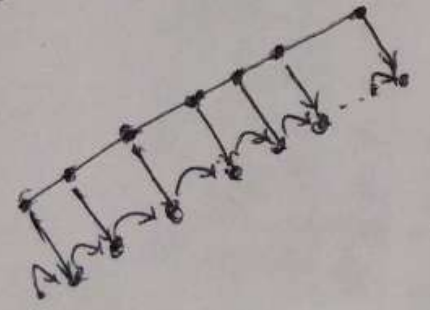
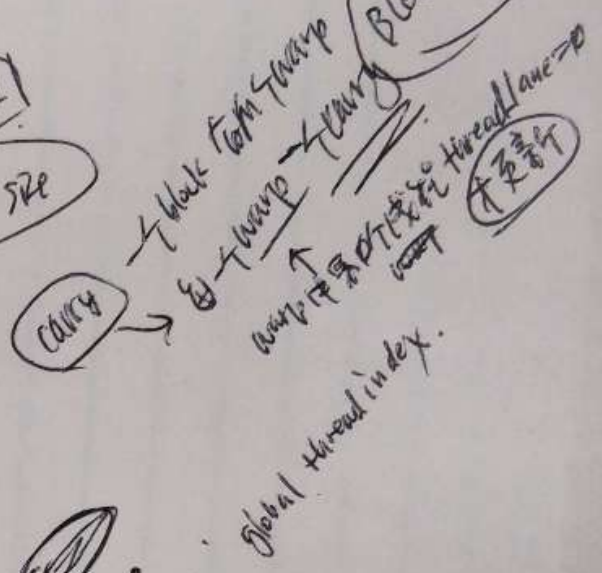
$sdata$: block thread shared memory

WARPS PER BLOCK = $\frac{\text{Block Size}}{\text{Warp Size}}$
 $= \frac{256}{32} = 8$



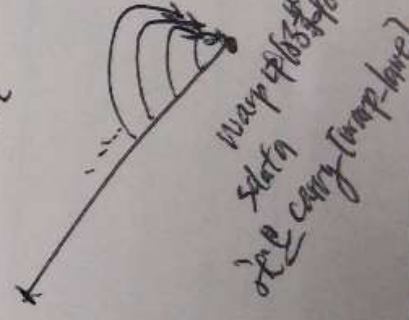
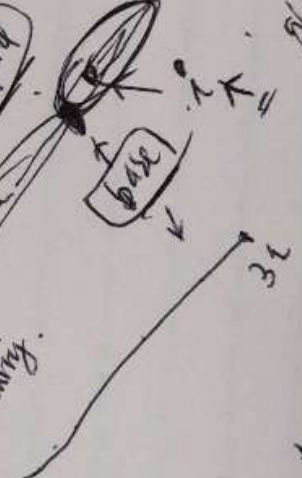
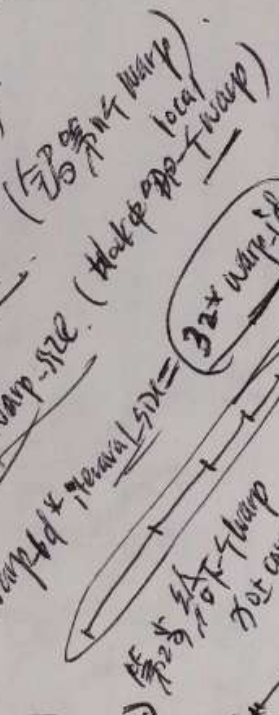
num_items = $\frac{\text{Warp Size}}{\text{Warp Size}} \times \text{num_warps}$
 $\text{Thread Size} = \text{Warp Size} \times \text{num_warps}$
 $+ 8$

per-warp size
 16 位 为 1
 16 位 为 2 位

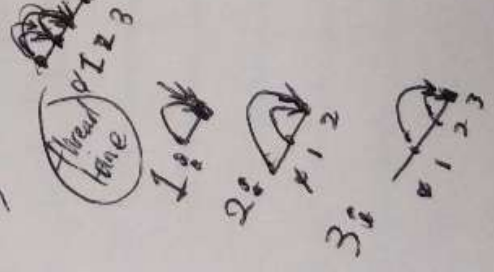
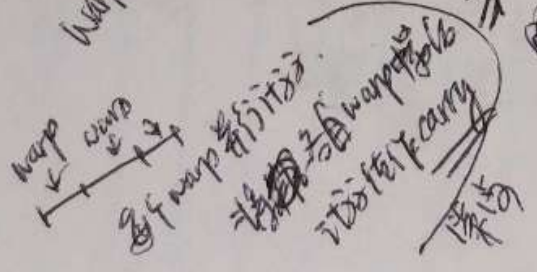
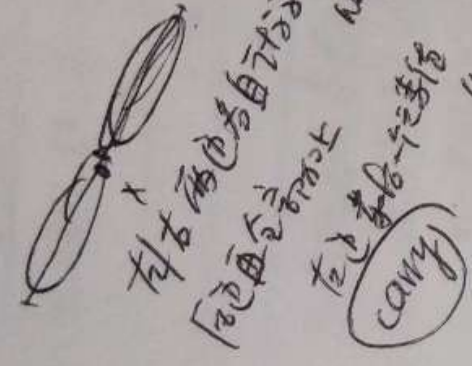


thread lane
 warp 中 第 几 个 线程

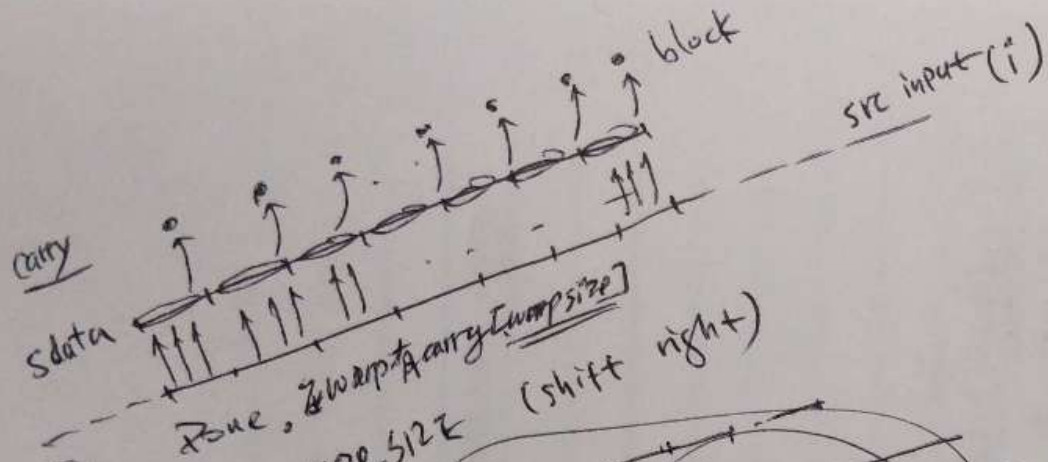
$\text{thread_idx} \times \text{warp_size}$
 $\text{Global Thread Id} / \text{warp_size}$
 $\text{warp_lane} = \text{thread_idx} \times \text{warp_size}$
 $\text{base} = \text{warp_id} \times \text{warp_size}$



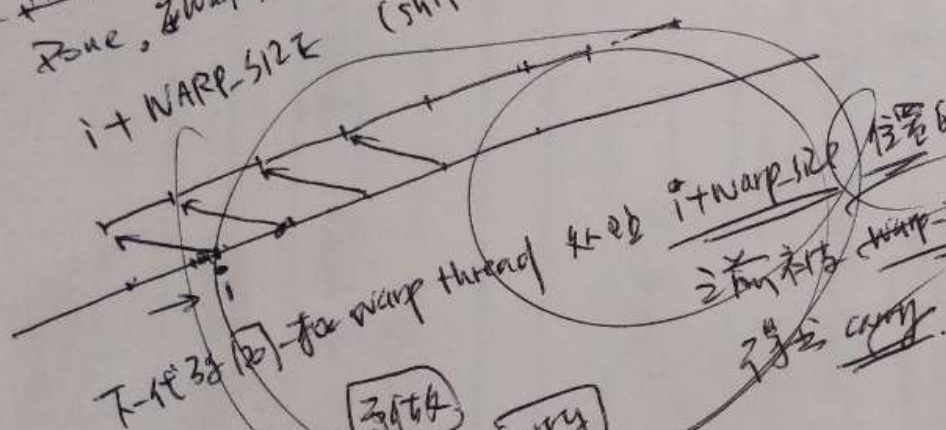
16 位 为 1



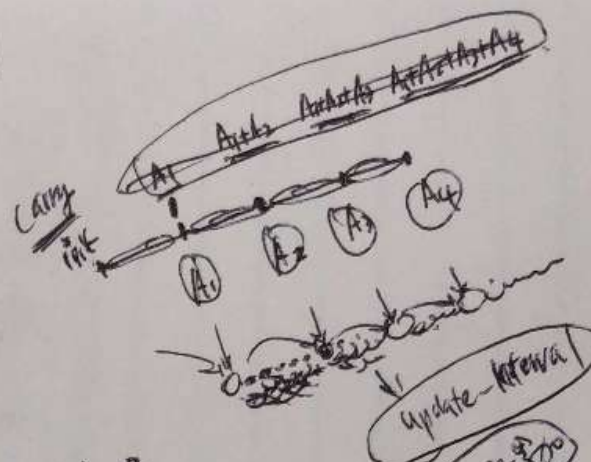
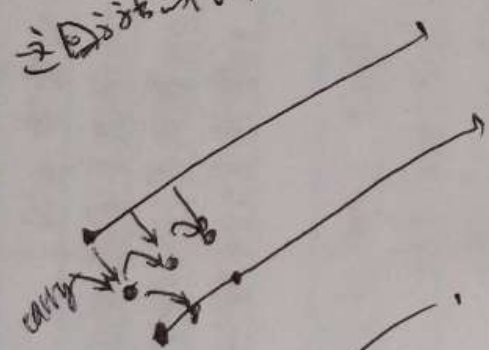
warp 中 第 几 个 线程
 16 位 为 1
 16 位 为 2 位



①
②



这一iteration 3次
但需要加上 carry
上一轮的 carry
这轮就 OK 了，但



① 先算 $i + i \times \text{warp size}$ carry
→ 同 dst 的 partial sum
② inclusive scan.

再算 inclusive scan.
 $A_1 \ A_2 \ \dots \ A_n$
→ $A_1 \ A_1+A_2 \ A_1+A_2+A_3 \ \dots \ A_1+A_2+\dots+A_n$
③ 这中间 warp interval
 $A_1 \ A_1+A_2 \ A_1+A_2+A_3$
↓ ↓ ↓
这中间 $A_1 + \dots$

sdata
 T0
 T1
 T2
 T3
 T4
 T5
 T8

T3在(做)时 T1已访问①
 故sdata[0]是
 sdata[0]+sdata[1]
 在1/2处

