CHAPTER 13

Sorting

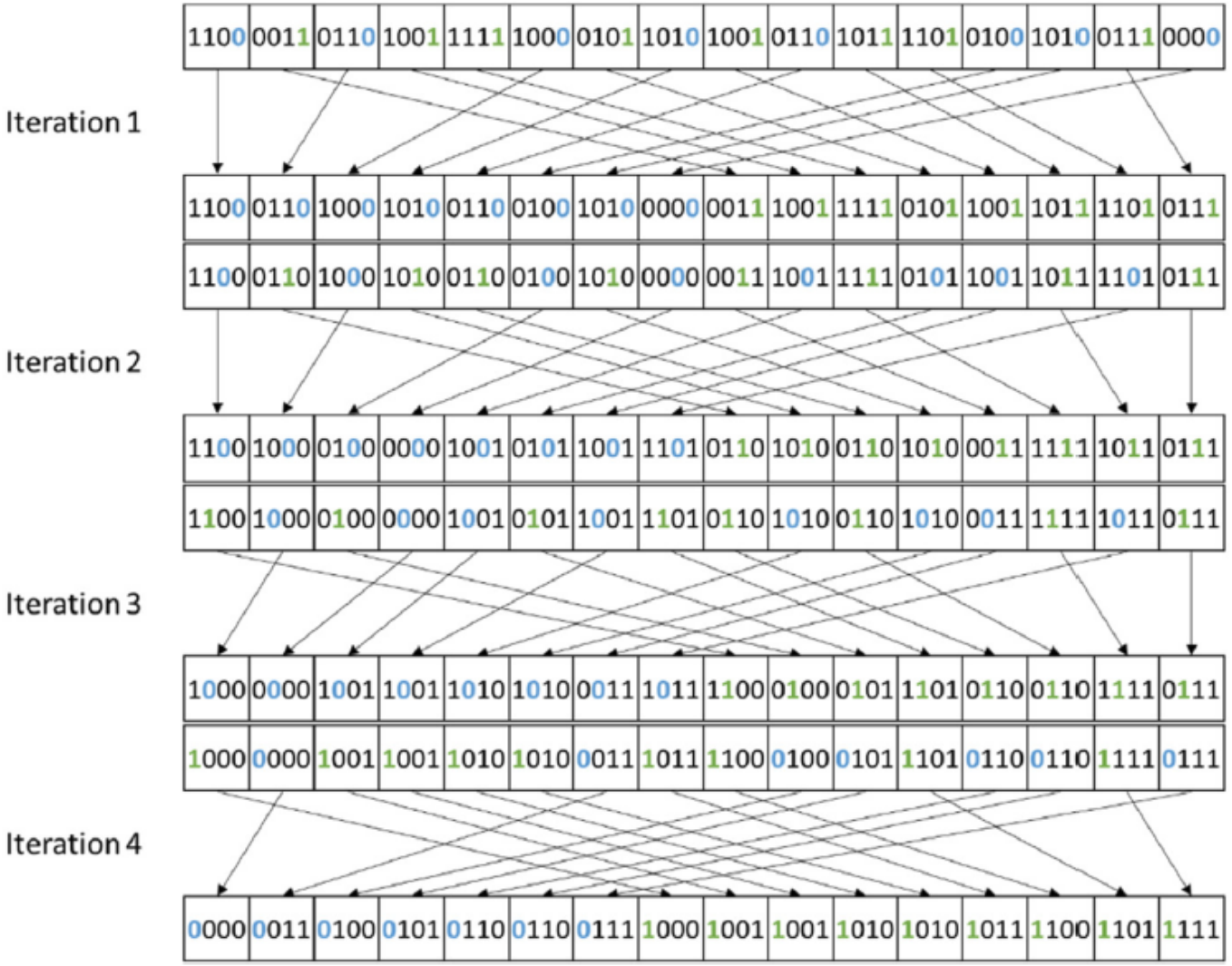


FIGURE 13.1

A radix sort example.

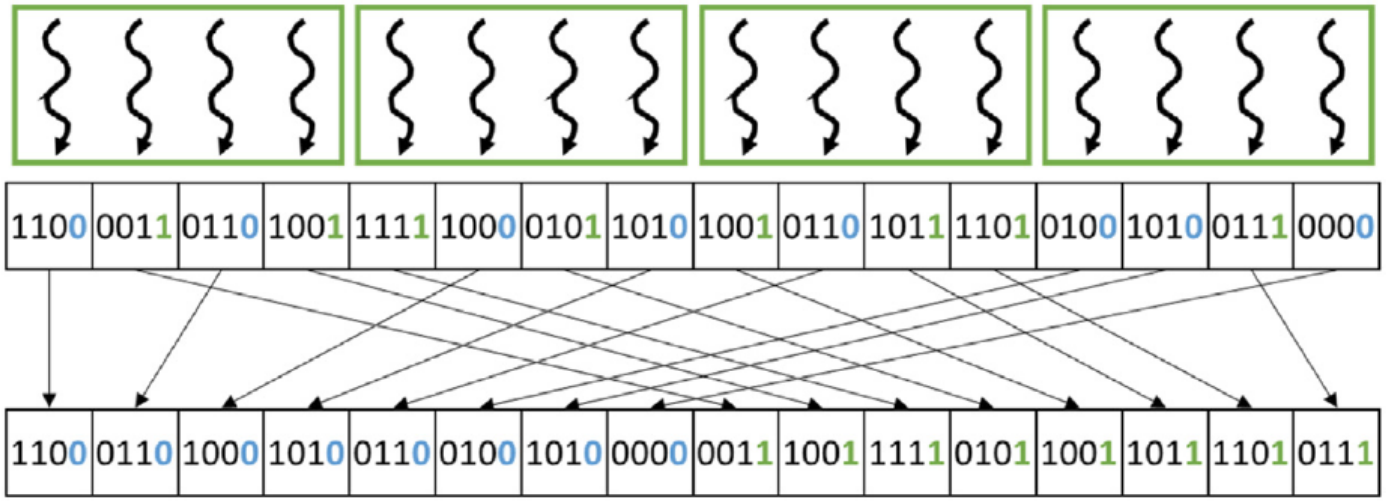


FIGURE 13.2

Parallelizing a radix sort iteration by assigning one input key to each thread.

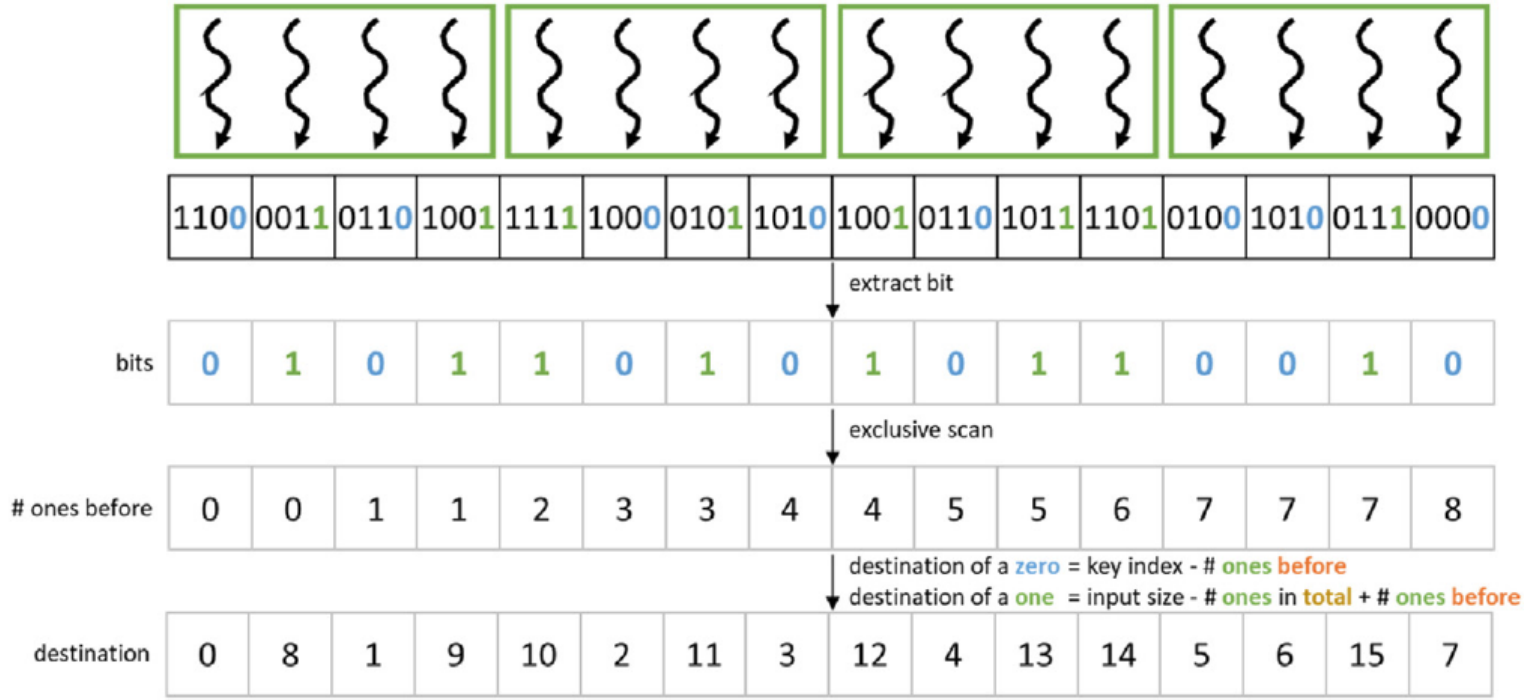


FIGURE 13.3

Finding the destination of each input key.

```
01  __global__ void radix_sort_iter(unsigned int* input, unsigned int* output,
02                       unsigned int* bits, unsigned int N, unsigned int iter) {
03      unsigned int i = blockIdx.x*blockDim.x + threadIdx.x;
04      unsigned int key, bit;
05      if(i < N) {
06          key = input[i];
07          bit = (key >> iter) & 1;
08          bits[i] = bit;
09      }
10      exclusiveScan(bits, N);
11      if(i < N) {
12          unsigned int numOnesBefore = bits[i];
13          unsigned int numOnesTotal = bits[N];
14          unsigned int dst = (bit == 0)?(i - numOnesBefore)
15                                       :(N - numOnesTotal - numOnesBefore);
16          output[dst] = key;
17      }
18  }
```

FIGURE 13.4

Radix sort iteration kernel code.

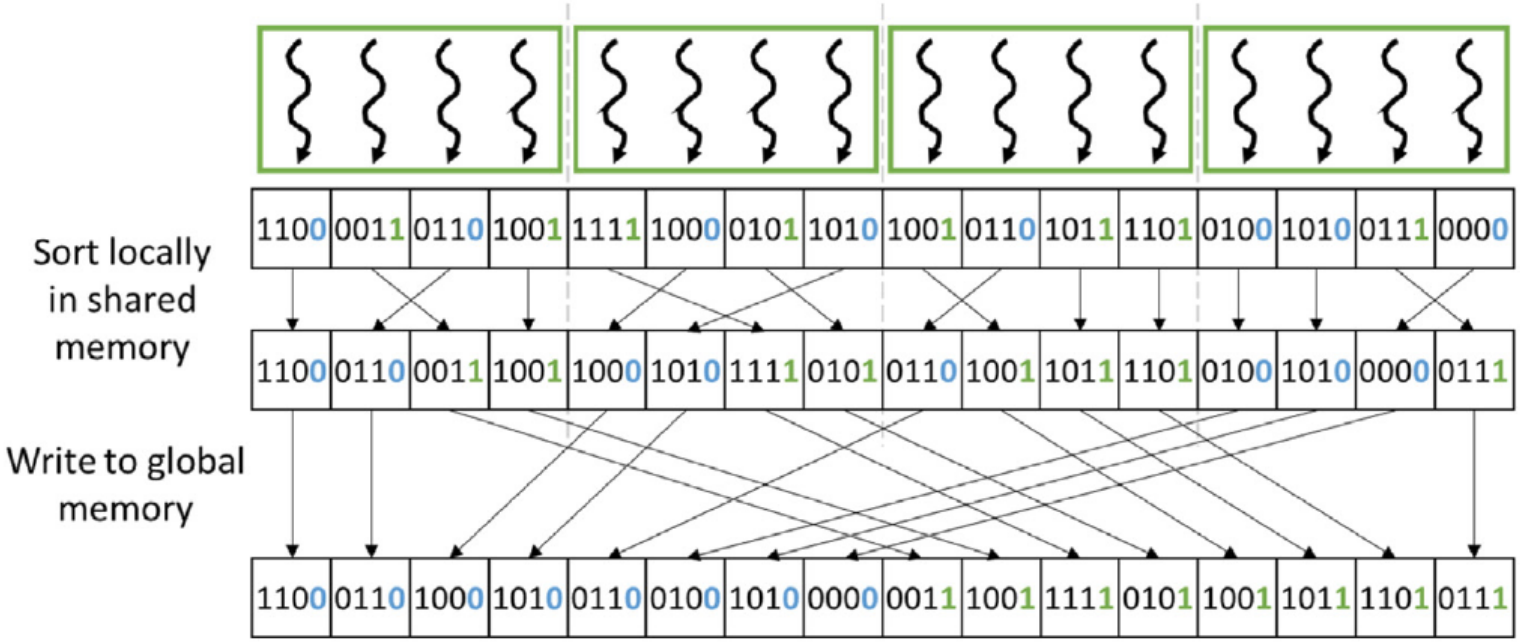


FIGURE 13.5

Optimizing for memory coalescing by sorting locally in shared memory before sorting into the global memory.

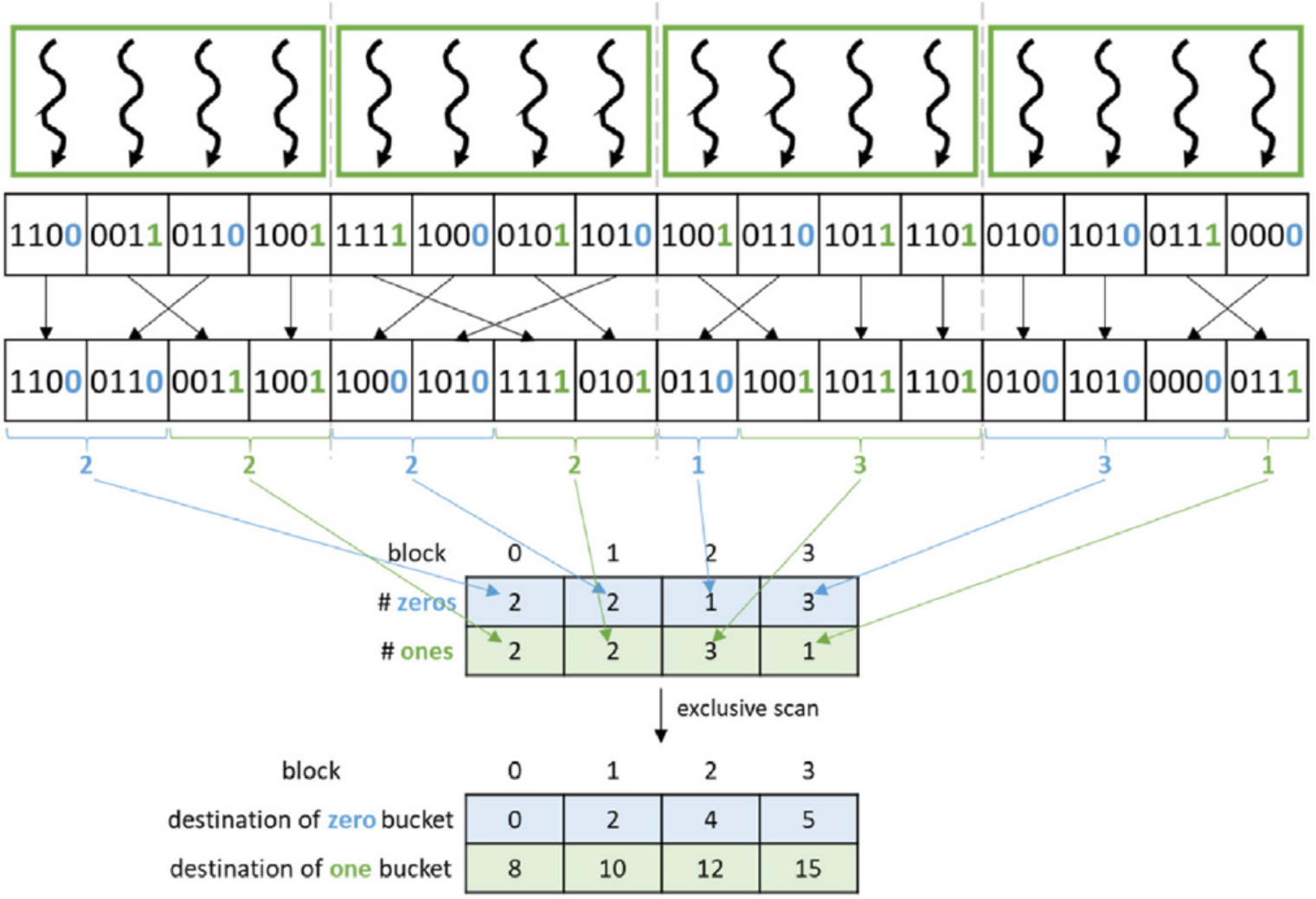


FIGURE 13.6

Finding the destination of each thread block's local buckets.

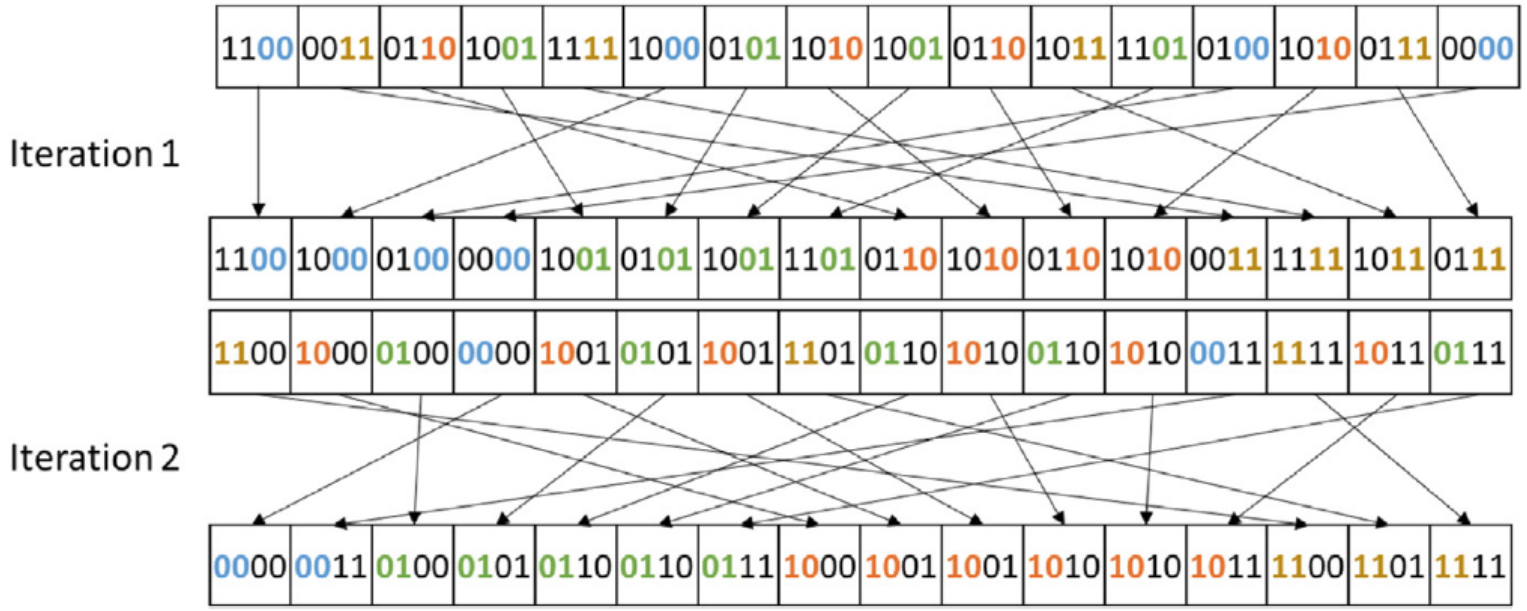


FIGURE 13.7

Radix sort example with 2-bit radix.

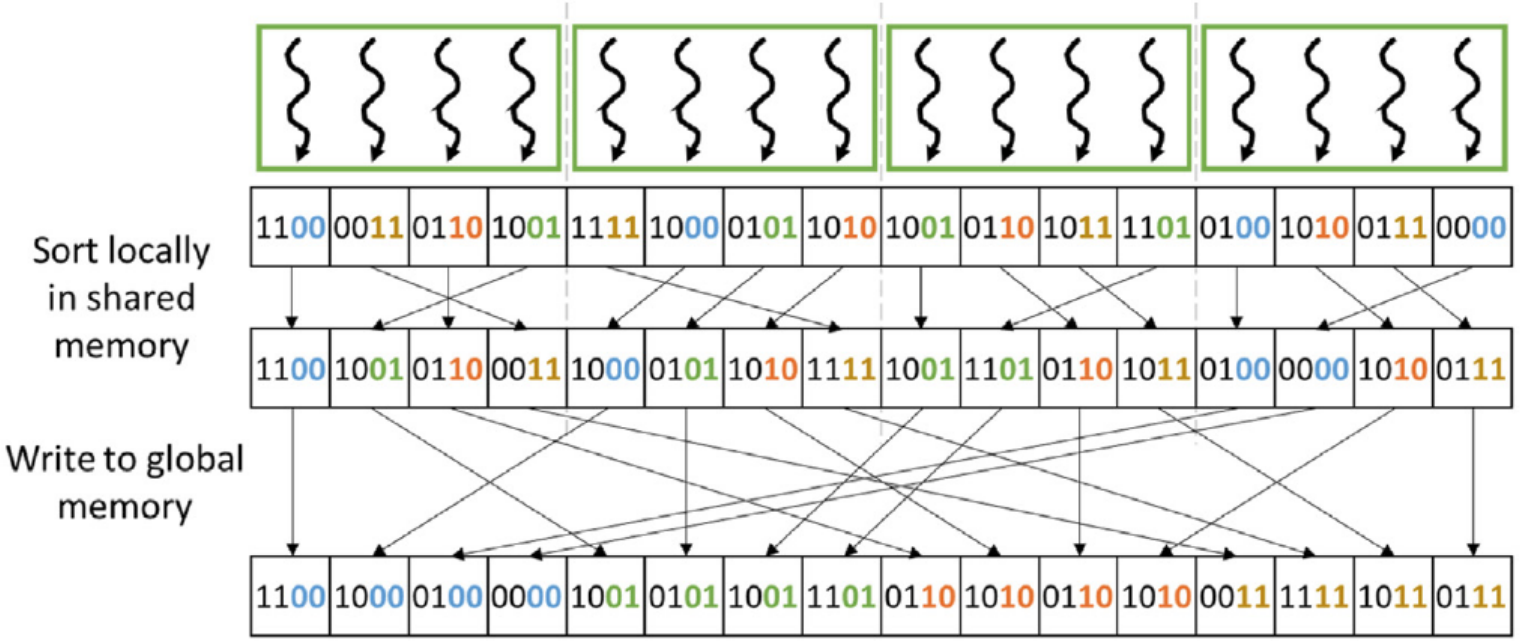


FIGURE 13.8

Parallelizing a radix sort iteration and optimizing it for memory coalescing using the shared memory for a 2-bit radix.

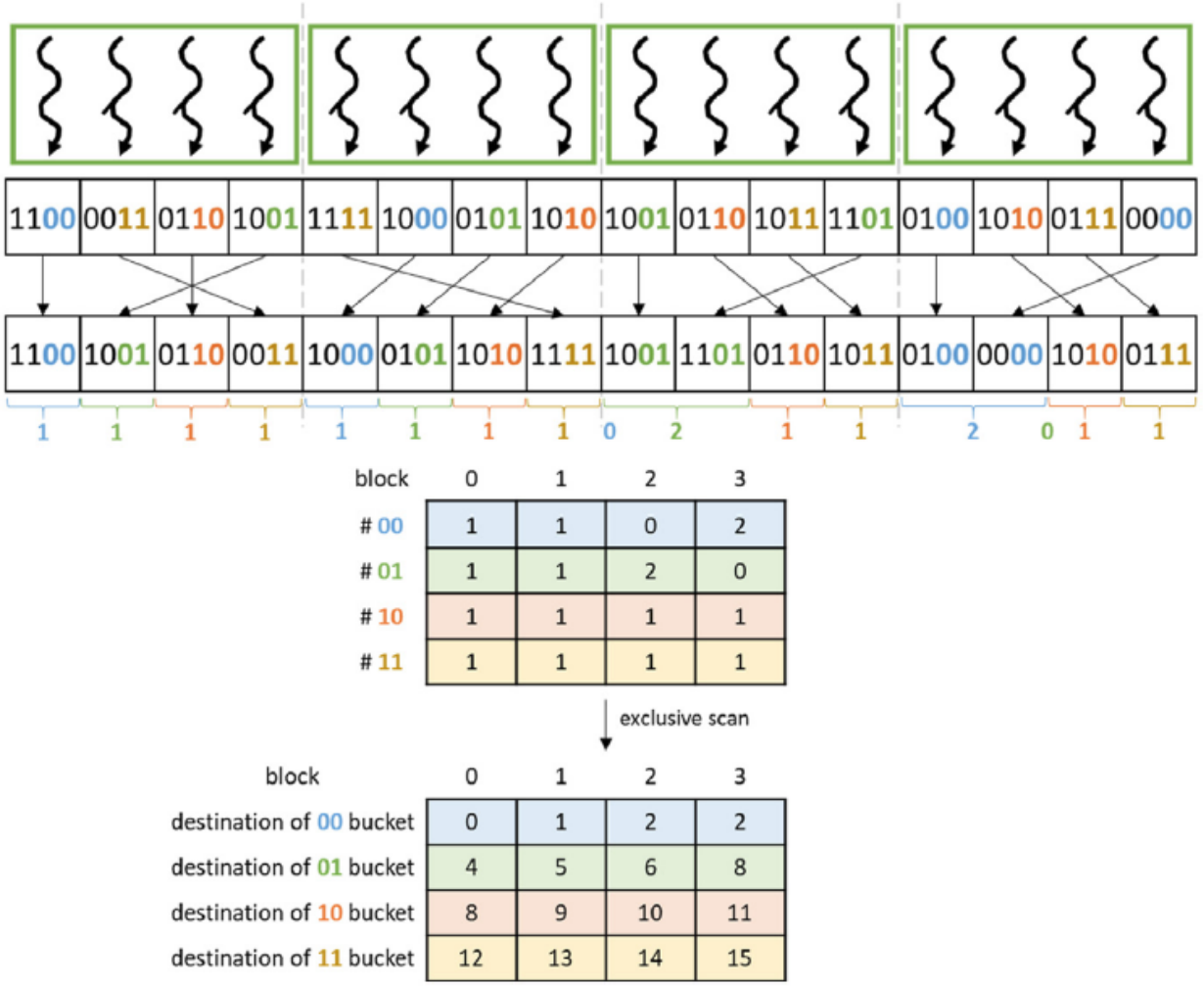


FIGURE 13.9

Finding the destination of each block's local buckets for a 2-bit radix.

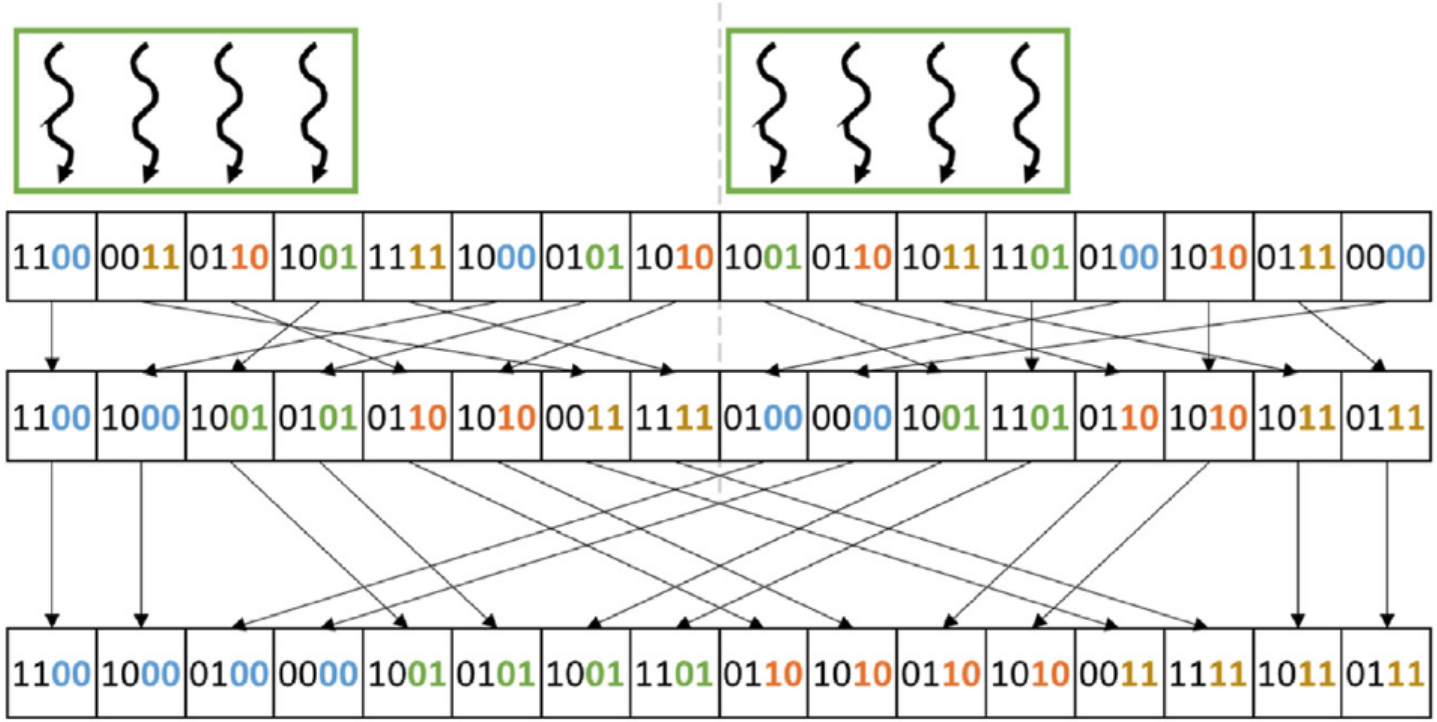


FIGURE 13.10

Radix sort for a 2-bit radix with thread coarsening to improve memory coalescing.

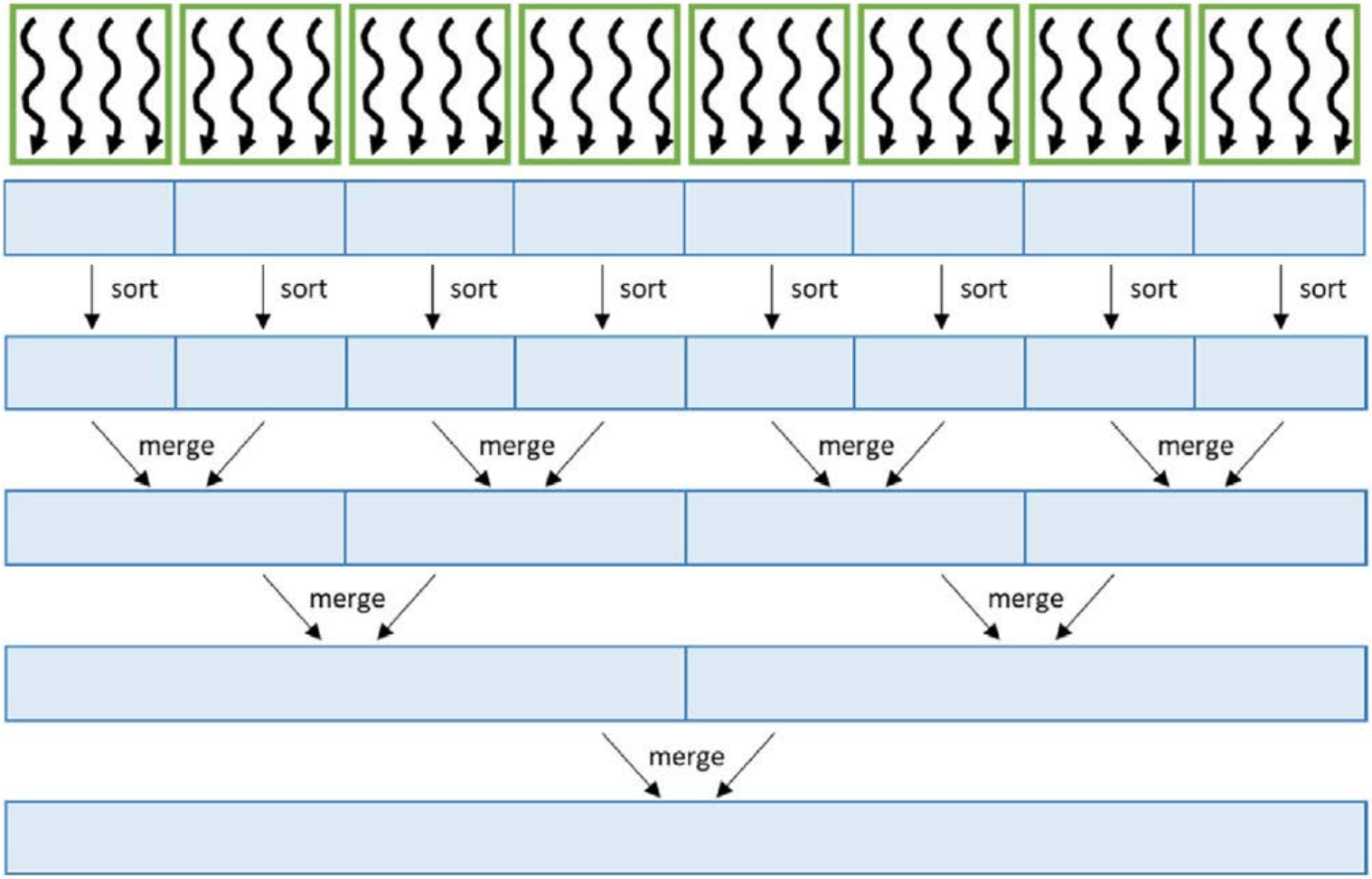
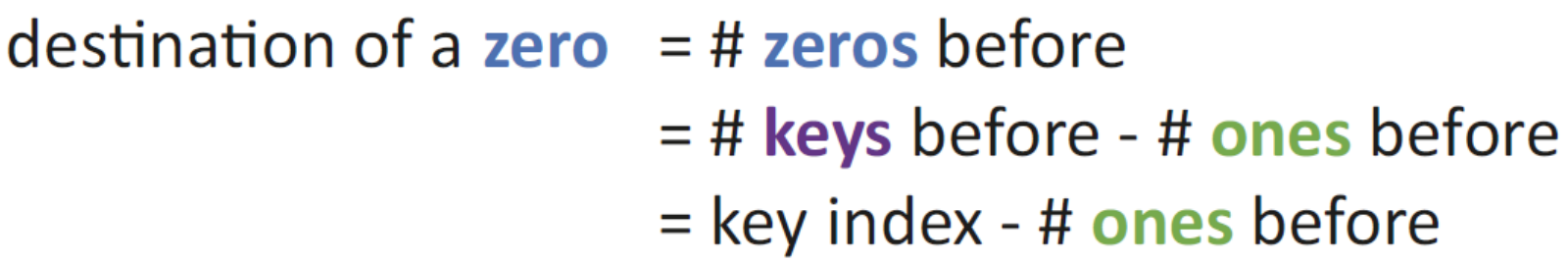


FIGURE 13.11

Parallelizing merge sort.



In-text figure 1

destination of a one = # zeros in total + # ones before
= (# keys in total - # ones in total) + # ones before
= input size - # ones in total + # ones before

In-text figure 2