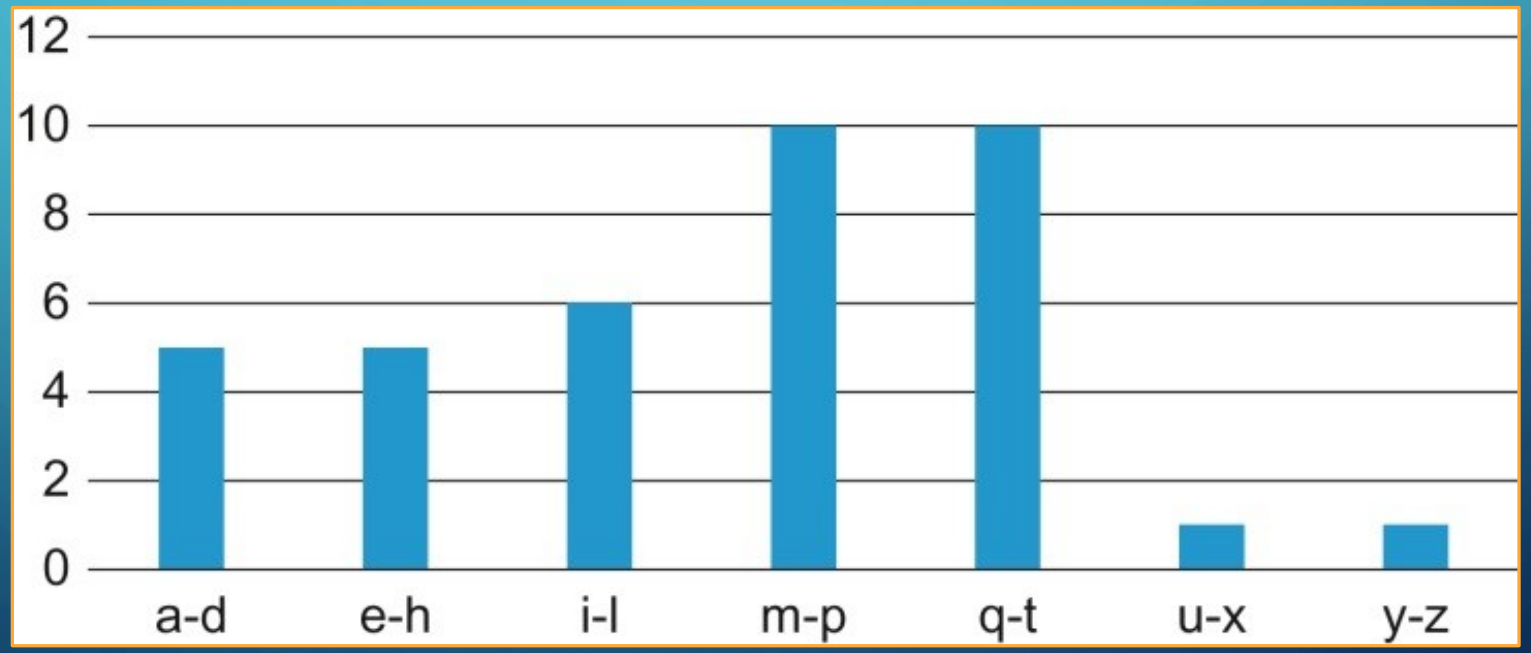
PARALLEL HISTOGRAM COMPUTATION

WHAT IS HISTOGRAM

- A histogram is a display of the frequency of data items in successive numerical intervals.
- In the most common form of histogram, the value intervals are plotted along the horizontal axis and the frequency of data items in each interval is represented as the height of a rectangle, or bar, rising from the horizontal axis.



WHAT IS HISTOGRAM

- Histograms of different types of object images, such as faces versus cars, tend to exhibit different shapes.
- By dividing an image into subareas and analyzing the histograms for these subareas, one can quickly identify the interesting subareas of an image that potentially contain the objects of interest.
- The process of computing histograms of image subareas is the basis of feature extraction in computer vision, where feature refers to patterns of interest in images.
- Credit card fraudulence detection and computer vision obviously meet this description. Other application domains with such needs include speech recognition, website purchase recommendations, and scientific data analysis such as correlating heavenly object movements in astrophysics.

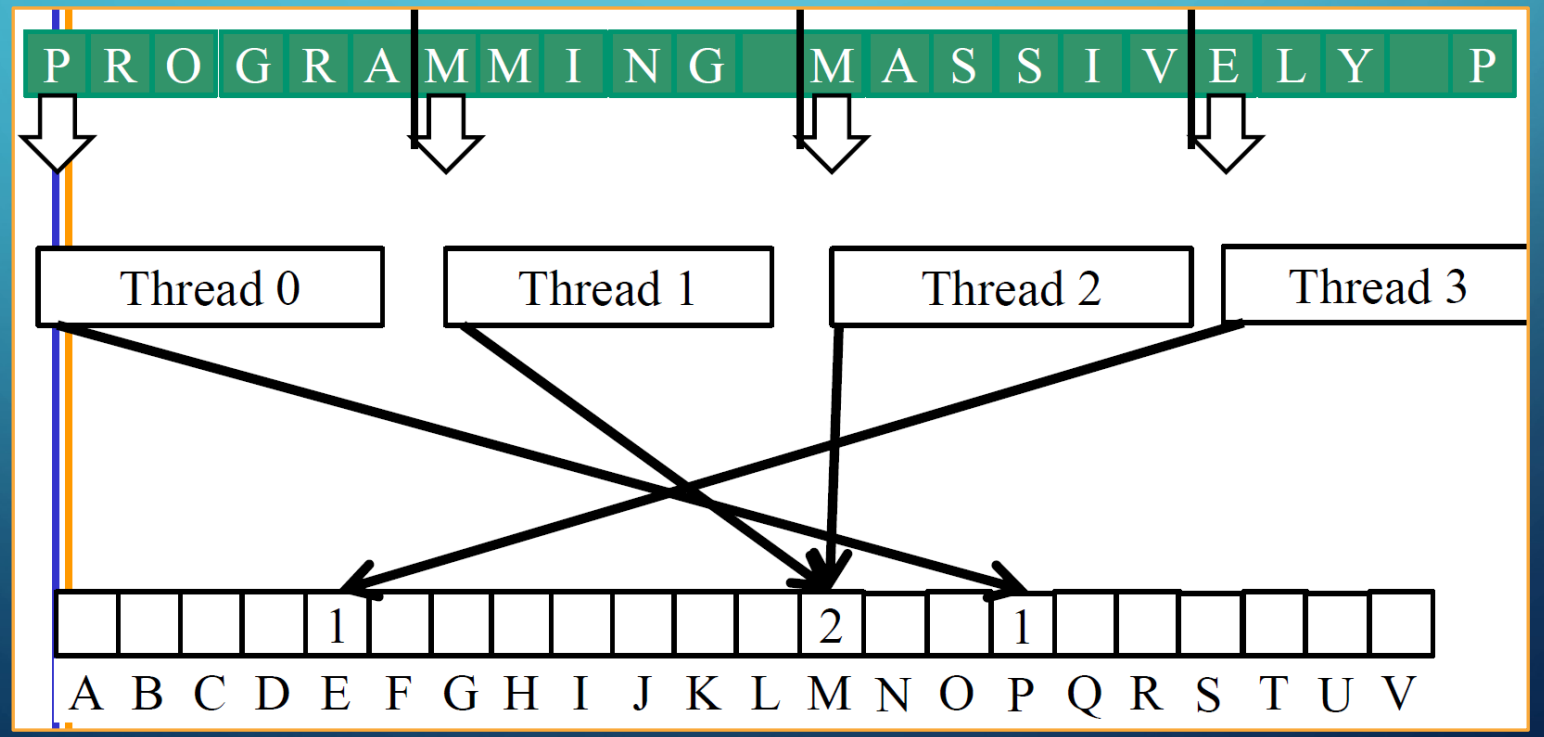
WHAT IS HISTOGRAM

- Histograms can be easily computed in a sequential manner, as shown below.

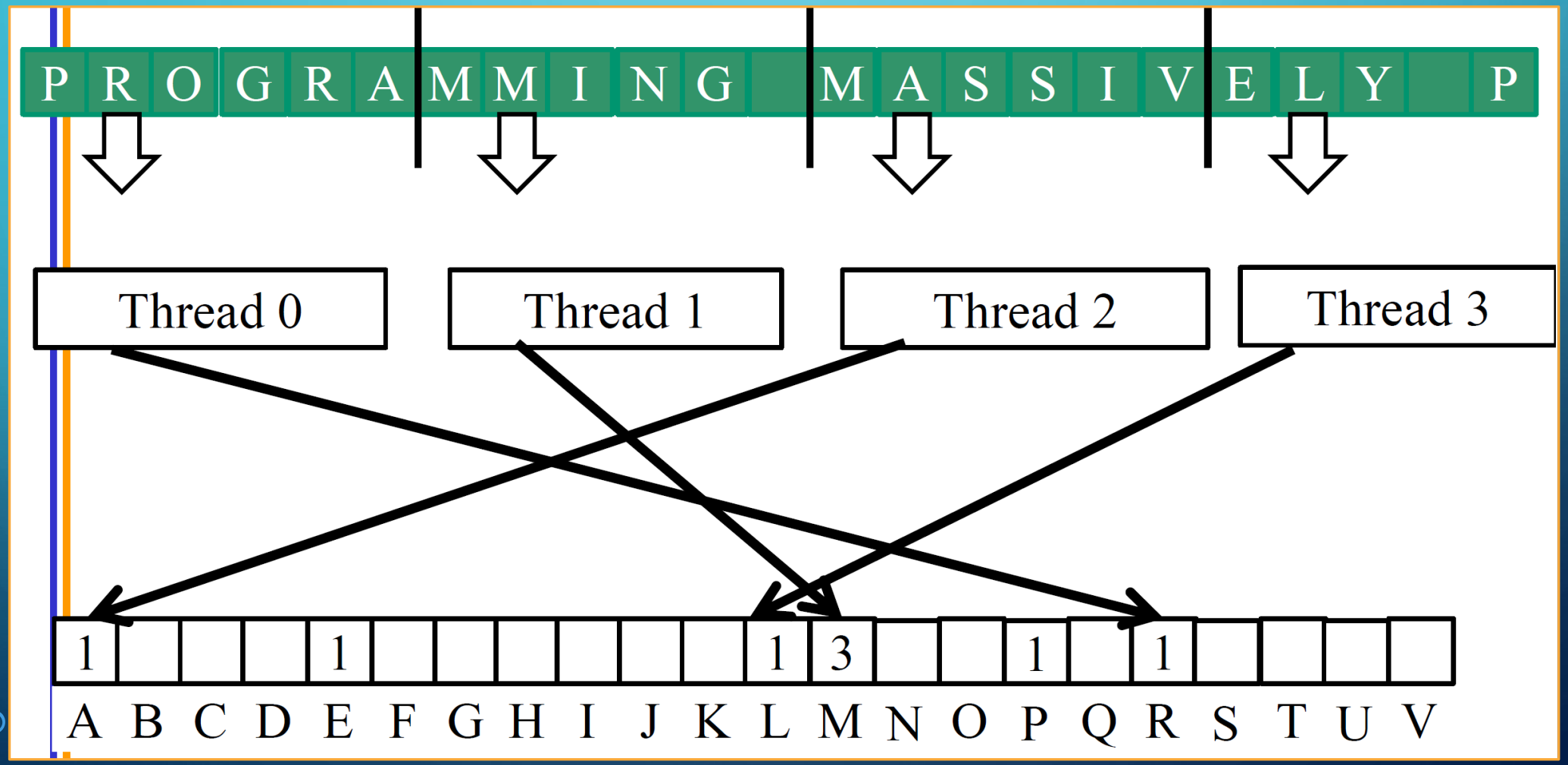
```
1.  sequential_Histogram(char *data, int length, int *histo) {
2.      for (int i = 0; i < length; i++) {
3.          int alphabet_position = data[i] – ‘a’;
4.          if (alphabet_position >= 0 && alphabet_position < 26) {
5.              histo[alphabet_position/4]++
6.          }
7.      }
8.  }
```

STRATEGY I

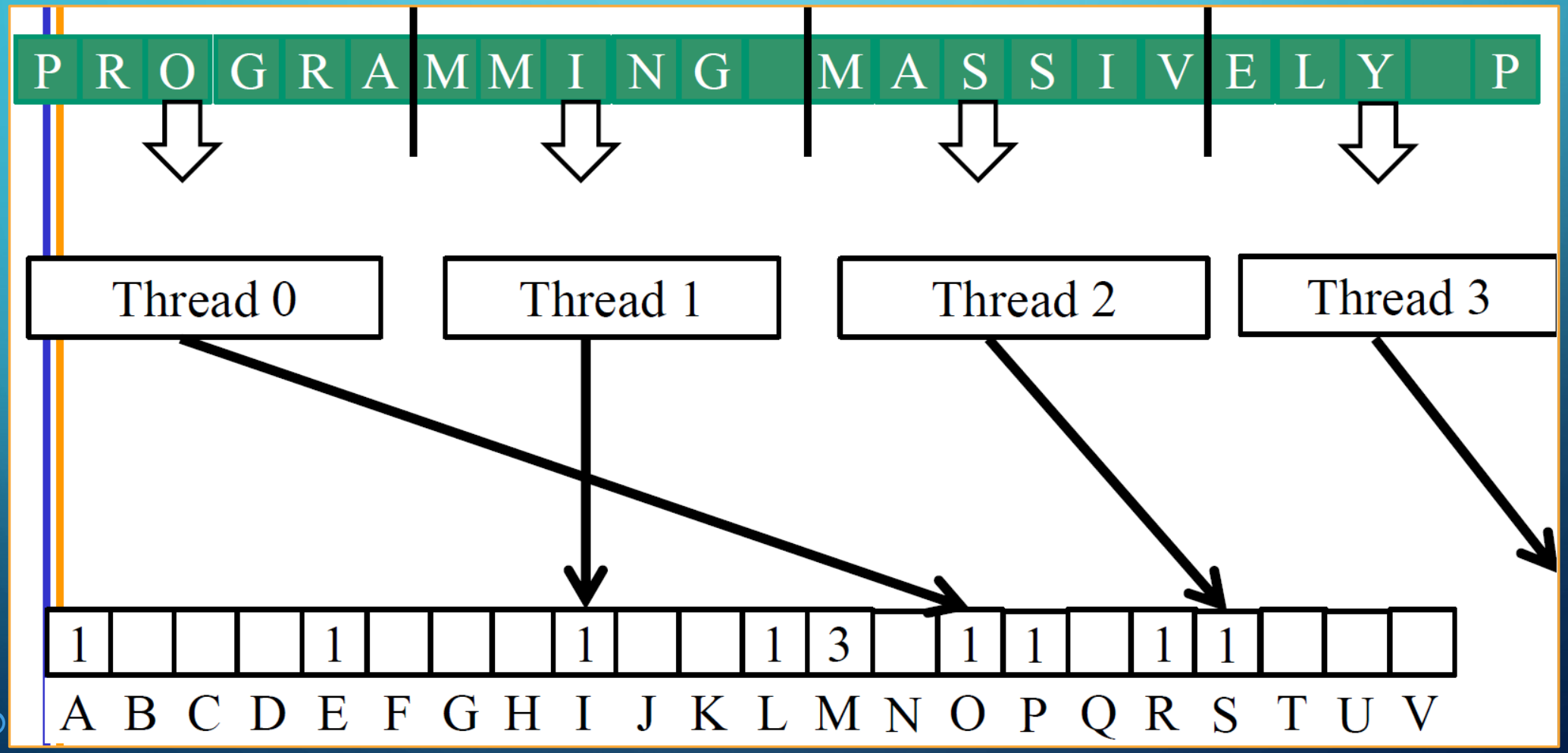
- A straightforward strategy for parallel histogram computation is dividing the input array into sections and having each thread process one of the sections.
- If we use P threads, each thread would be doing approximately 1/P of the original work.
- We assume that P = 4 and each thread processes a section of 6 characters.



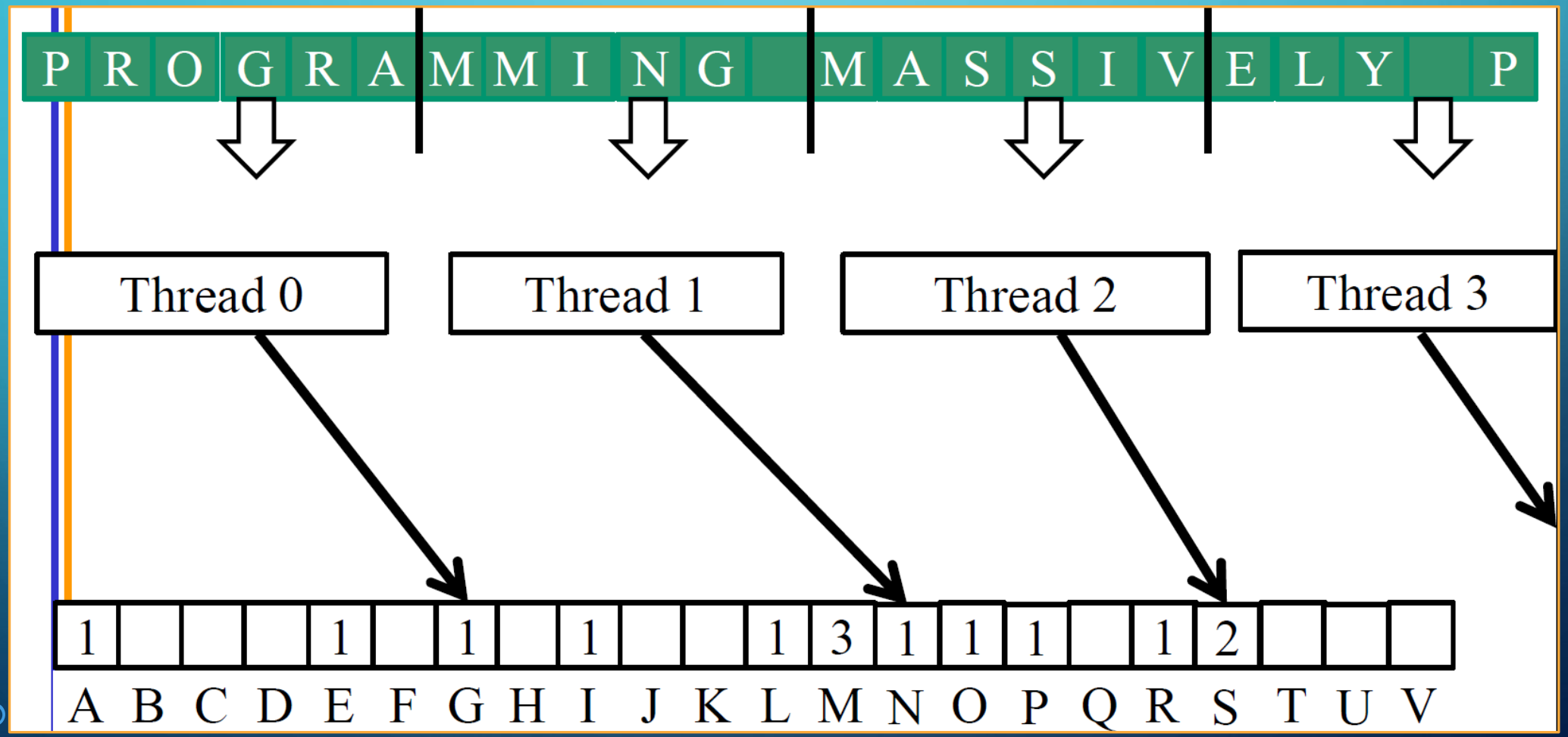
ITERATION #2


P R O G R A M M I N G M A S S I V E L Y P
Thread 0
Thread 1
Thread 2
Thread 3
1 1 1 3 1 1
A B C D E F G H I J K L M N O P Q R S T U V

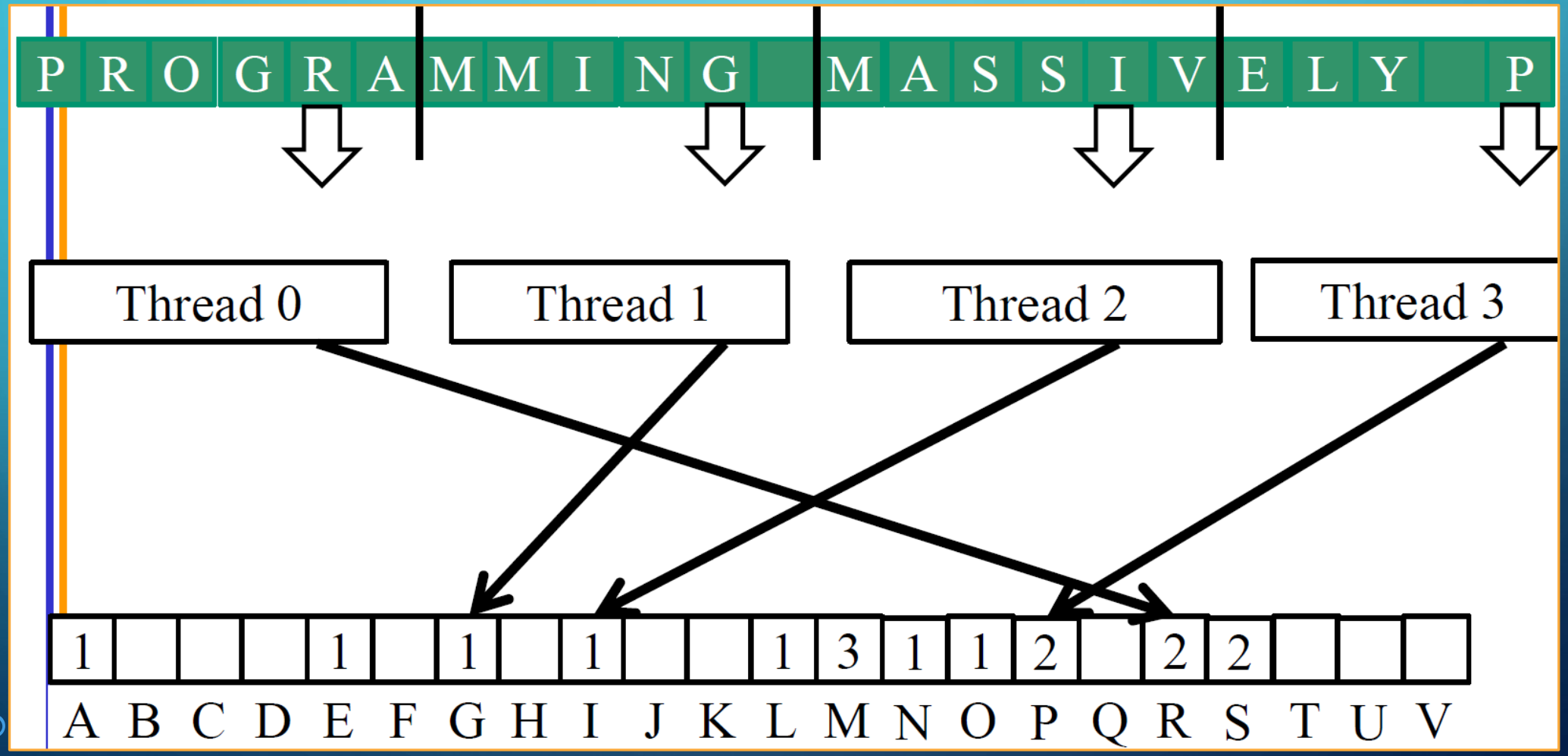
ITERATION #3


P R O G R A M M I N G M A S S I V E L Y P
Thread 0
Thread 1
Thread 2
Thread 3
1 1 1 1 3 1 1 1 1
A B C D E F G H I J K L M N O P Q R S T U V

ITERATION #4


P R O G R A M M I N G M A S S I V E L Y P
Thread 0
Thread 1
Thread 2
Thread 3
1 1 1 1 1 3 1 1 1 1 2
A B C D E F G H I J K L M N O P Q R S T U V

ITERATION #5


P R O G R A M M I N G M A S S I V E L Y P
Thread 0
Thread 1
Thread 2
Thread 3
1 1 1 1 1 3 1 1 2 2 2
A B C D E F G H I J K L M N O P Q R S T U V

RACE CONDITION

Time	Thread 1	Thread 2
1	(0) Old ← histo[x]	
2	(1) New ← Old + 1	
3	(1) histo[x] ← New	
4		(1) Old ← histo[x]
5		(2) New ← Old + 1
6		(2) histo[x] ← New

(A)

Time	Thread 1	Thread 2
1	(0) Old ← histo[x]	
2	(1) New ← Old + 1	
3		(0) Old ← histo[x]
4	(1) histo[x] ← New	
5		(1) New ← Old + 1
6		(1) histo[x] ← New

(B)

RACE CONDITION

Time	Thread 1	Thread 2
1		(0) Old ← histo[x]
2		(1) New ← Old + 1
3		(1) histo[x] ← New
4	(1) Old ← histo[x]	
5	(2) New ← Old + 1	
6	(2) histo[x] ← New	

(A)

Time	Thread 1	Thread 2
1		(0) Old ← histo[x]
2		(1) New ← Old + 1
3	(0) Old ← histo[x]	
4		(1) histo[x] ← New
5	(1) New ← Old + 1	
6	(1) histo[x] ← New	

(B)

A CUDA KERNEL FOR CALCULATION HISTOGRAM BASED ON STRATEGY I

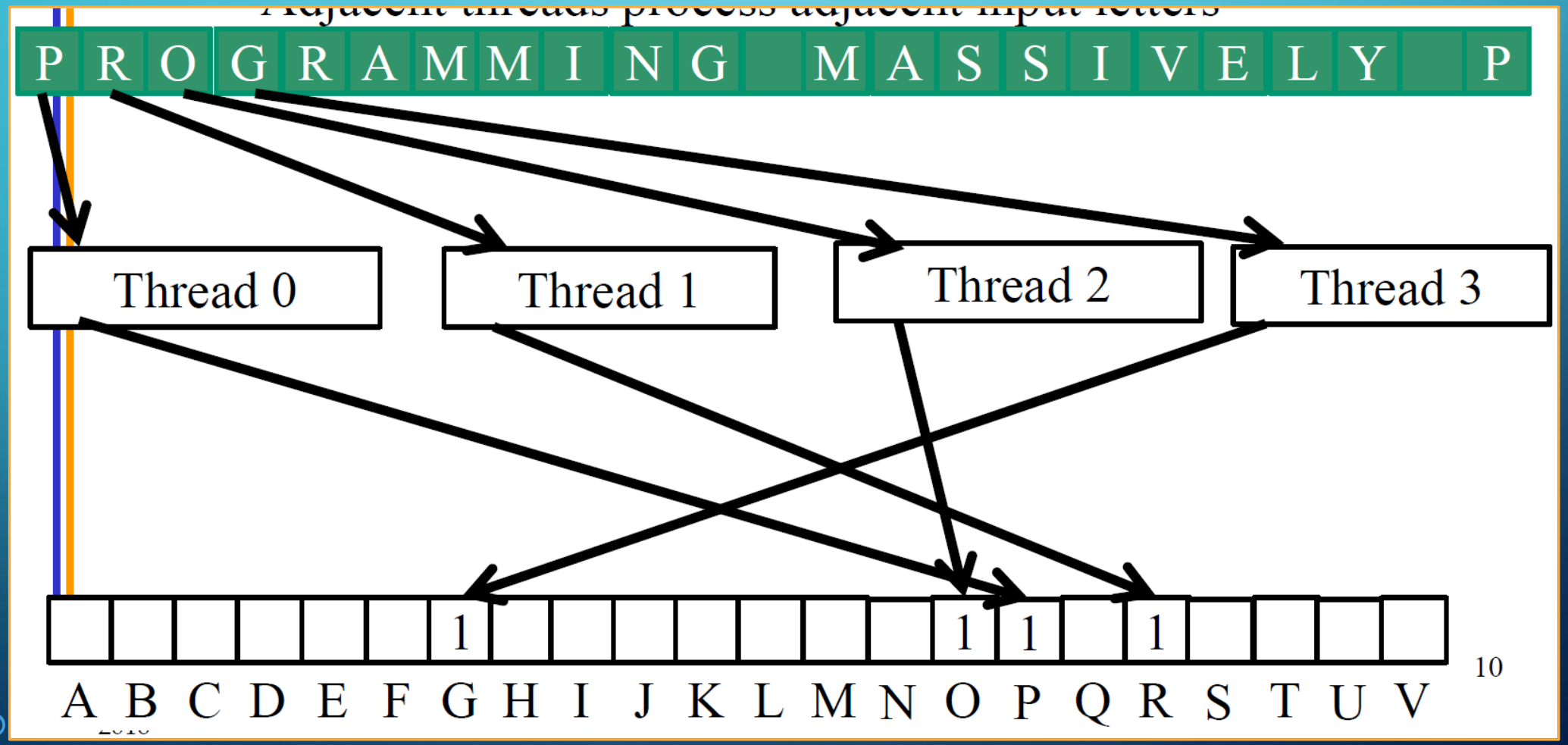
```
__global__ void histo_kernel(unsigned char *buffer, long size, unsigned int *histo)
{
1.   int i = threadIdx.x + blockIdx.x * blockDim.x;
2.   int section_size = (size-1)  / (blockDim.x * gridDim.x) +1;
3.   int start = i*section_size;

 // All threads handle blockDim.x * gridDim.x
   // consecutive elements
4.   for (k = 0; k < section_size; k++) {
5.    if (start+k < size) {
6.      int alphabet_position = buffer[start+k] – ‘a’;
7.      if (alphabet_position >= 0 && alpha_position < 26) atomicAdd(&(histo[alphabet_position/4]), 1);
       }
     }
}
```

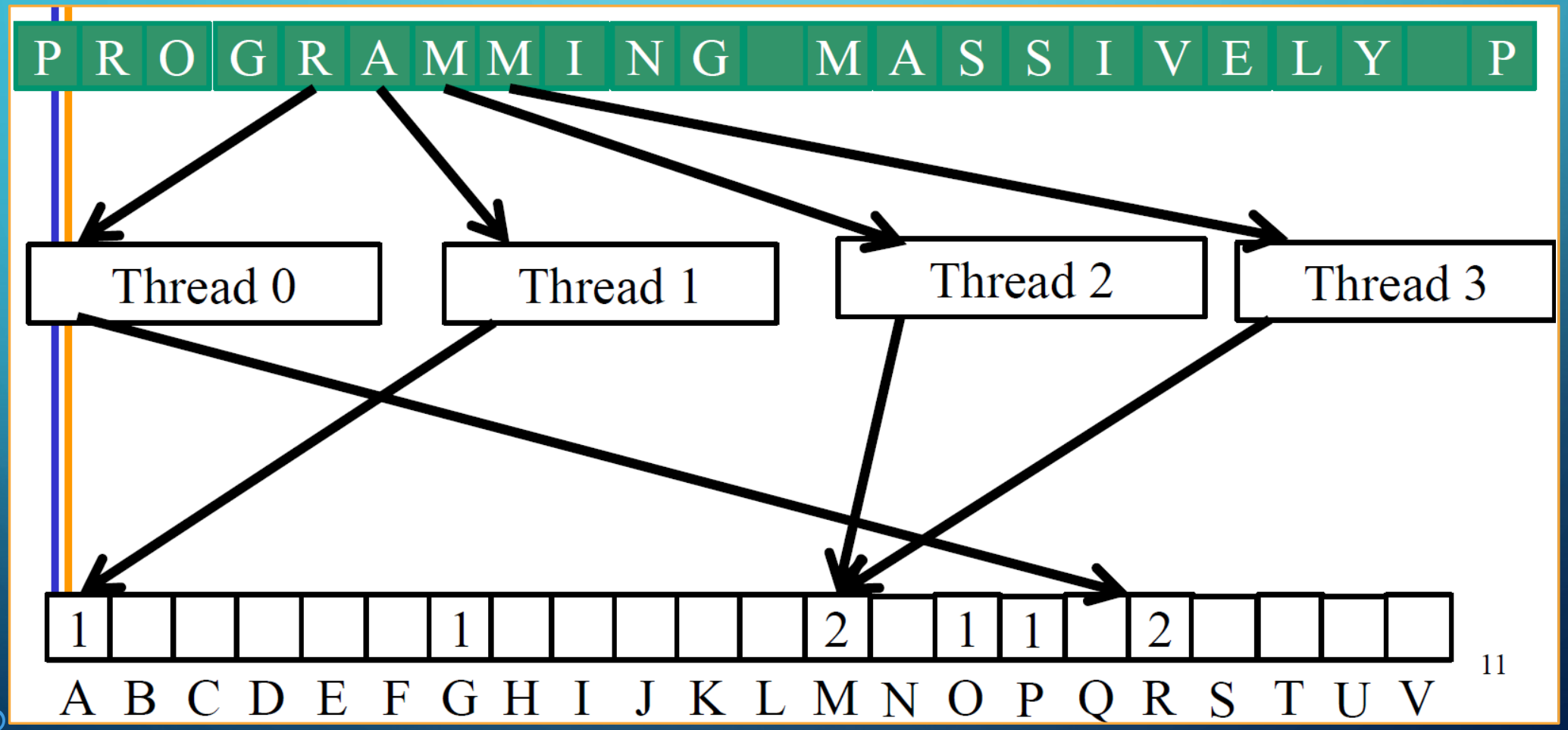
BLOCK VERSUS INTERLEAVED PARTITIONING

- In Strategy I, we partition the elements of buffer[] into sections of continuous elements, or blocks, and assign each block to a thread.
- This partitioning strategy is often referred to as block partitioning.
- Partitioning data into continuous blocks is an intuitive and conceptually simple approach.
- On a CPU, where parallel execution typically involves a small number of threads, block partitioning is often the best performing strategy since the sequential access pattern by each thread makes good use of cache lines.

STRATEGY II - ITERATION #1



ITERATION #2


P R O G R A M M I N G M A S S I V E L Y P
Thread 0
Thread 1
Thread 2
Thread 3
1 1 2 1 1 2
A B C D E F G H I J K L M N O P Q R S T U V
11

A CUDA KERNEL FOR CALCULATING HISTOGRAM BASED ON STRATEGY II

```
__global__ void histo_kernel(unsigned char *buffer, long size, unsigned int *histo)
{

1.  unsigned int tid = threadIdx.x + blockIdx.x * blockDim.x;

// All threads handle blockDim.x * gridDim.x consecutive elements in each iteration
2.   for (unsigned int i = tid;  i < size; i += blockDim.x*gridDim.x ) {
3.       int alphabet_position = buffer[i] – ‘a’;
4.       if (alphabet_position >= 0 && alpha_position < 26) atomicAdd(&(histo[alphabet_position/4]), 1);
     }
}
```

REFERENCES

- https://wiki.illinois.edu/wiki/display/ECE408/Class+Schedule
- Kirk, Hwu: Programming Massively Parallel Processors, 3rd Edition