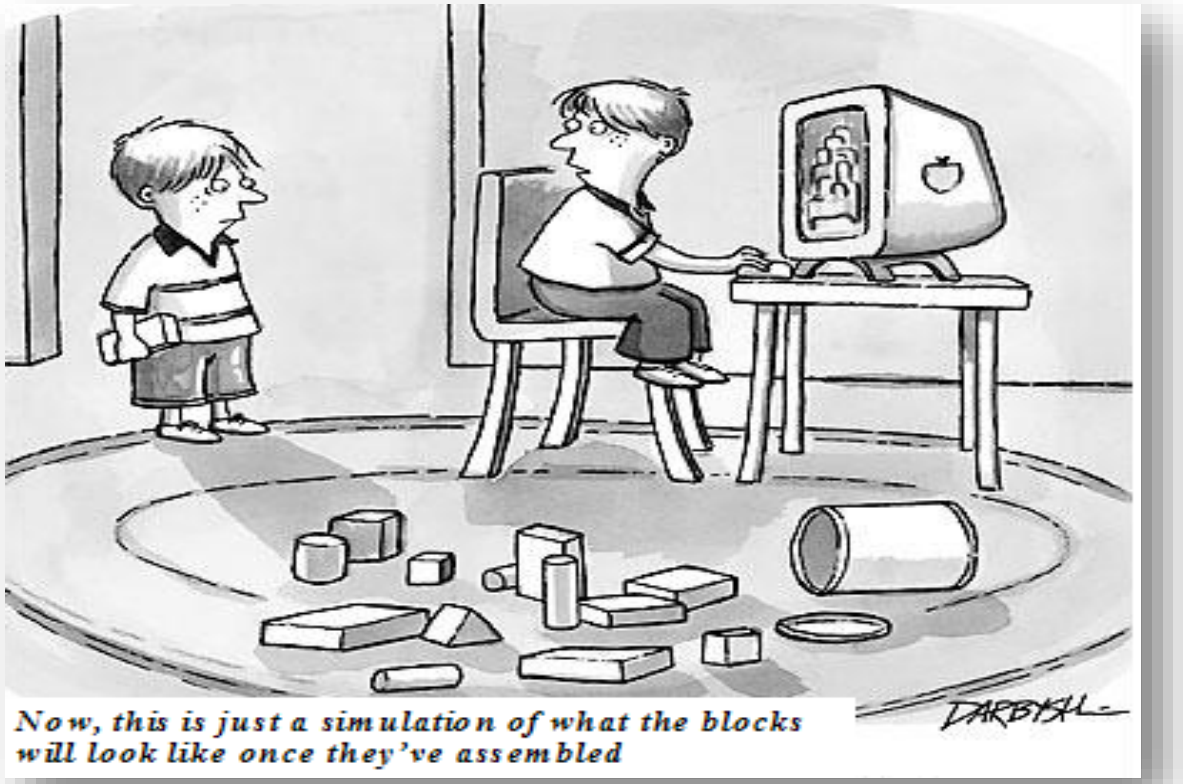
ECE569
Module 32



- Communication Patterns

Basic Efficiency Rules

- Expose enough parallelism
- Develop algorithms with a data parallel mindset
- Maximize locality of global memory accesses
- Exploit per-block shared memory as scratchpad
  - (registers > shared memory > global memory)
- Minimize divergence of execution within blocks

How do threads work together?

- **Communication**
  - Threads may need to
    - Read from the same input location
    - Write to the same output location
    - Exchange partial results
- **Critical Question**
  - How to map tasks and memory together?
    - Tasks = Threads in CUDA
    - Memory = Communication medium

Parallel Communication Patterns

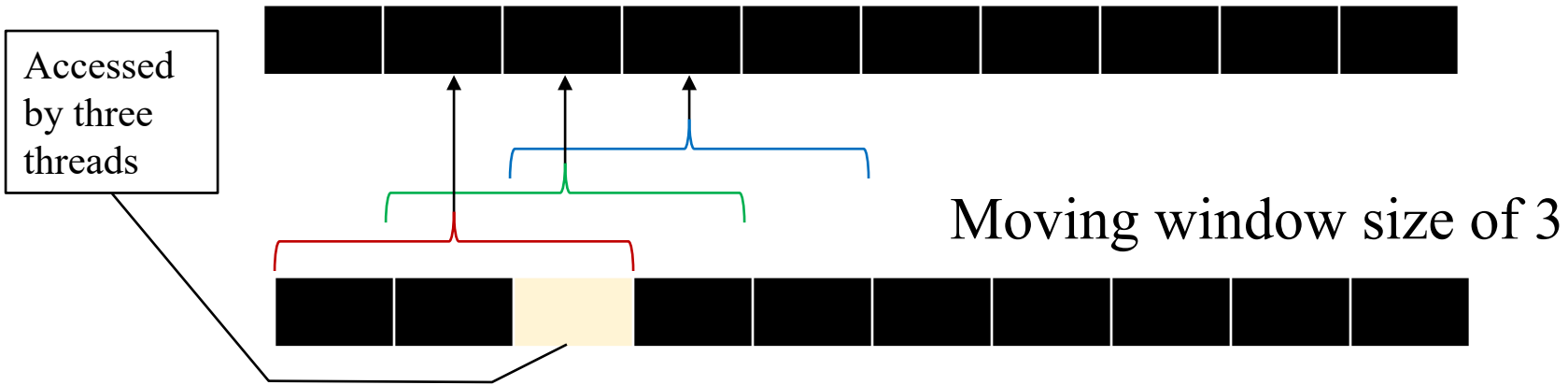
- Map and Transpose
- Gather
- Scatter
- Stencil
- Reduce
- Scan and Sort

Parallel Communication Patterns: MAP

- Many data elements
  - Pixels in an image
- Apply the same function on each data
- Each task reads from and writes to a specific place in memory
  - 1-to-1 correspondence between input and output
  - Easily expressed in CUDA
- **Which of these can be solved by using map?**
  - Sorting an input array, sum up elements in an array, Compute average of an input array

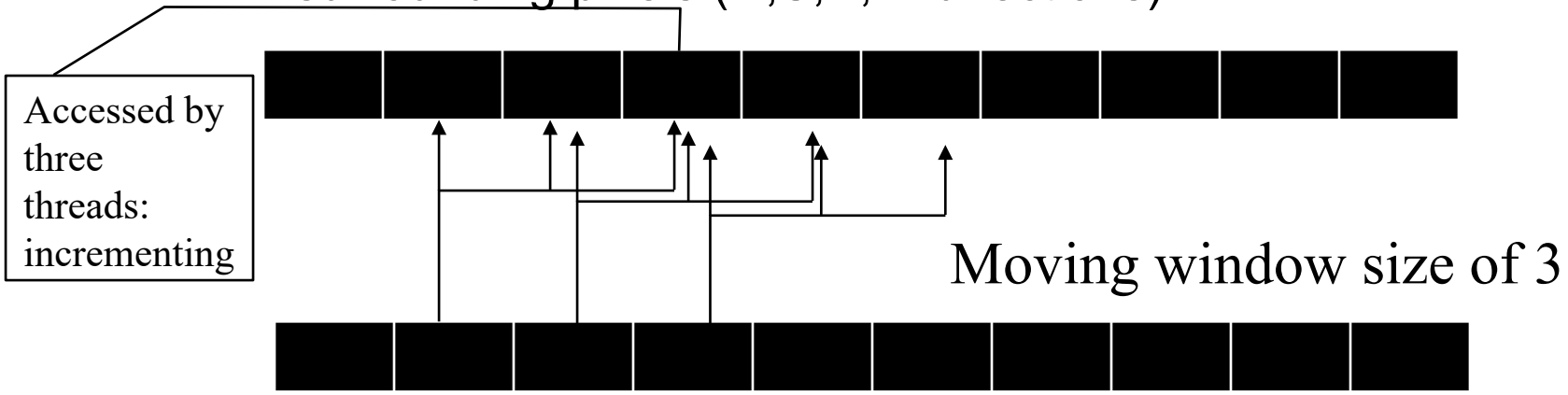
Parallel Communication Patterns: Gather

- Each calculation gathers input data elements together to compute an output result
  - For example: Blurring an image
- Function: Compute the average of three elements in a moving window manner and store the average
- Each thread reads three locations from the memory and writes into a single place



Parallel Communication Patterns: Scatter

- Each parallel task needs to write its results **in a different place** or **in multiple places**
  - Threads scatter the results over memory
  - **Challenge:** multiple threads try to write into the same address at the same time!
- Rather than having each thread read 3 neighboring elements, each thread reads one element but **adds** 1/3 of it's element's value to 3 neighboring elements
  - In 2D similar to image blurring, each pixel updates its 4 surrounding pixels (N,S,E,W directions)



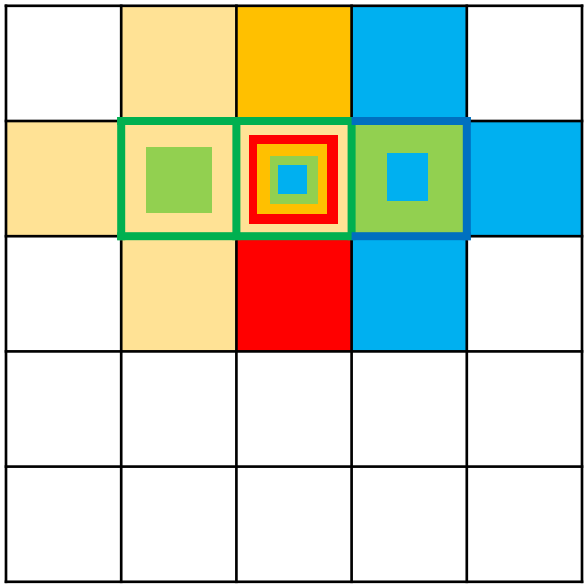
Parallel Communication Patterns

- **Given a list of basketball players**
  - Name
  - Height
  - Rank in height (tallest to shortest)
- **Write each player's record into its location in a sorted list**
- **Is this Map, Scatter or Gather?**

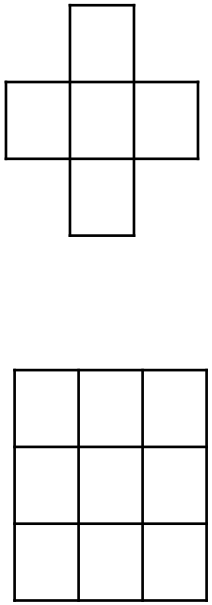
Parallel Communication Patterns: Stencil

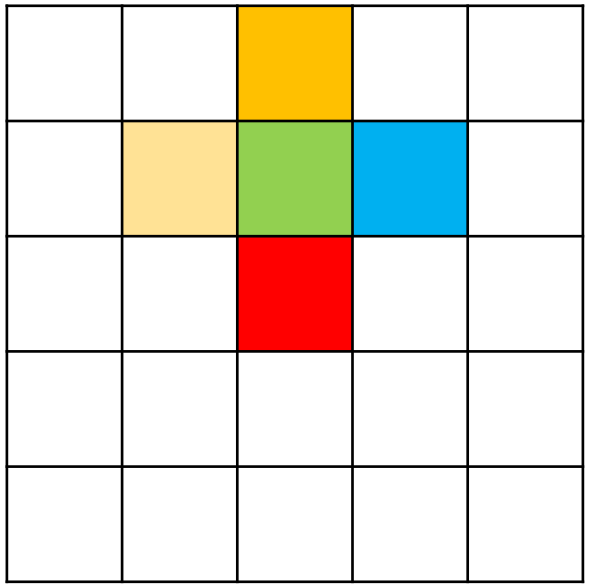
- Each thread **updates each element** of an array element using neighboring array elements in a fixed pattern
  - Pattern is called as stencil
- Data Reuse

How many times will a given input value be read?



Read





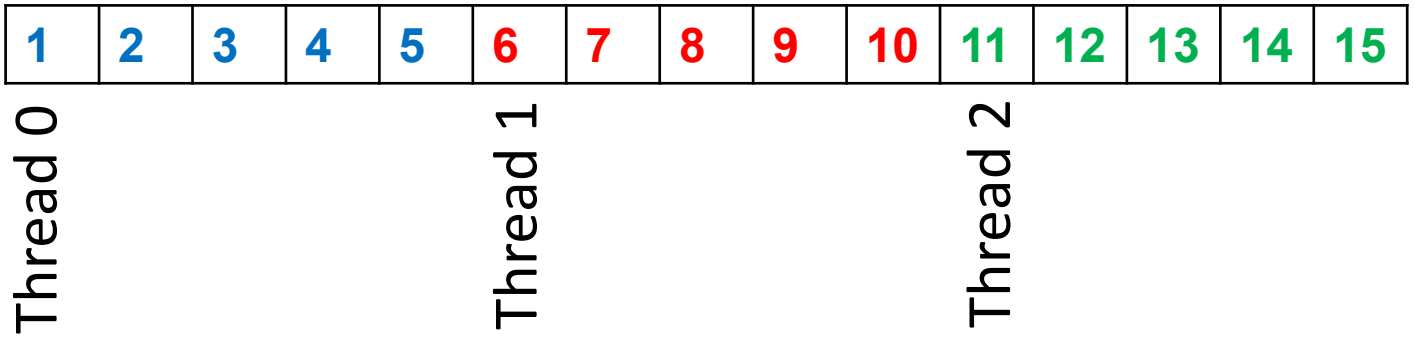
Write

Assumption: Threads write their output to a different array to avoid synchronization issues

Parallel Communication Patterns: Transpose

– Row major order to column major order

Thread 0	1	2	3	4	5
Thread 1	6	7	8	9	10
Thread 2	11	12	13	14	15



Parallel Communication Patterns: Transpose

– Row major order to column major order

Thread 0	1	2	3	4	5
Thread 1	6	7	8	9	10
Thread 2	11	12	13	14	15

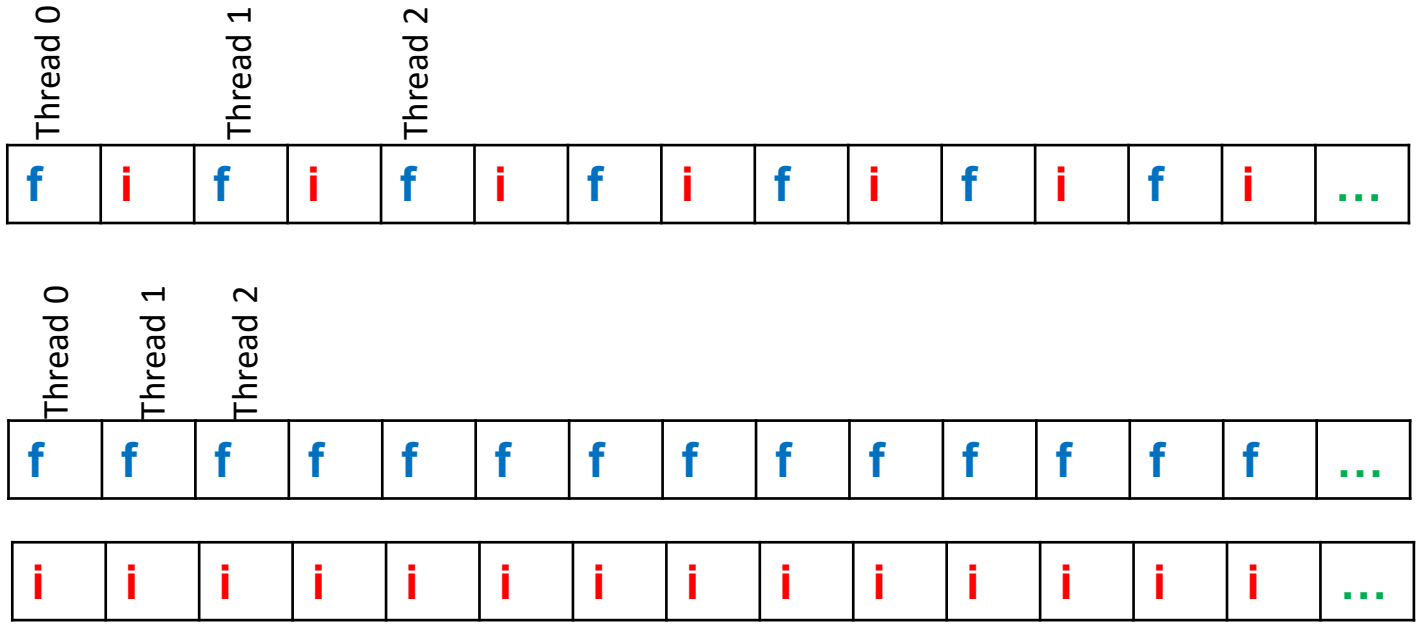
1	6	11
2	7	12
3	8	13
4	9	14
5	10	15

Thread 0	Thread 1	Thread 2												
1	6	11	2	7	12	3	8	13	4	9	14	5	10	15

Parallel Communication Patterns: Transpose

– If we operate on “f” exclusively, it is better to transpose!

```
struct foo {
      float f;
      int i;
};
foo array[1000]; \\ array of structure
```



Structure of arrays

Exercise

- **Label each line by pattern**

```
float out[], in[];
int i = threadIdx.x;
int j=thredIdx.y;
const float pi=3.14;

out[i] = pi*in[i]

out[i+j*128] = in[j+i*128];

if (i%2){
out[i-1] += pi*in[i]; out[i+1] += pi*in[i];


 out[i] = (in[i]+in[i-1]+in[i+1])*pi/3.0;
}
```

A. Map
B. Gather
C. Scatter
D. Stencil
E. Transpose

Parallel Communication Patterns

- Map and Transpose
  - One-to-one
- Gather
  - Many-to-one
- Scatter
  - One-to-many
- Stencil
  - Specialized gather
  - Several to one
- Reduce (Next Topic)
  - All-to-one
- Scan and Sort
  - All-to-all