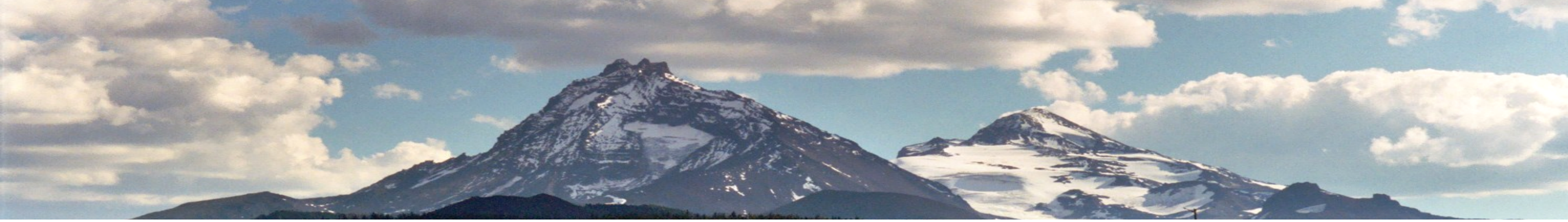


Stencil Pattern

Parallel Computing

CIS 410/510

Department of Computer and Information Science

UNIVERSITY OF OREGON

Table of Contents

Table of Contents

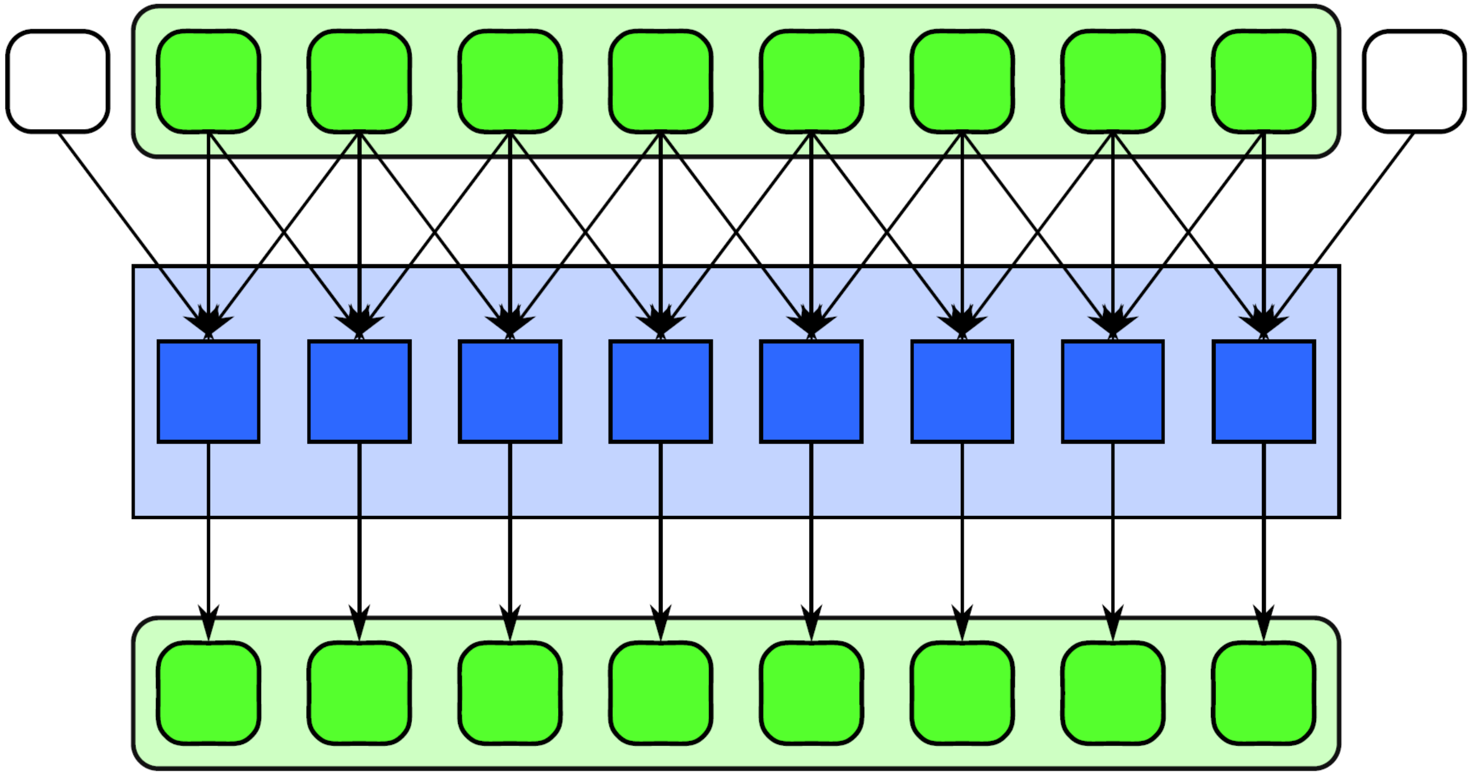
- What is the stencil pattern?
- Implementing stencil with shift
- Stencil and cache optimizations
- Stencil and communication optimizations
- Recurrence

What is the stencil pattern?

- **Stencil**: A map where each output depends on a "neighborhood" of inputs. These inputs are a set of fixed offsets relative to the output position.

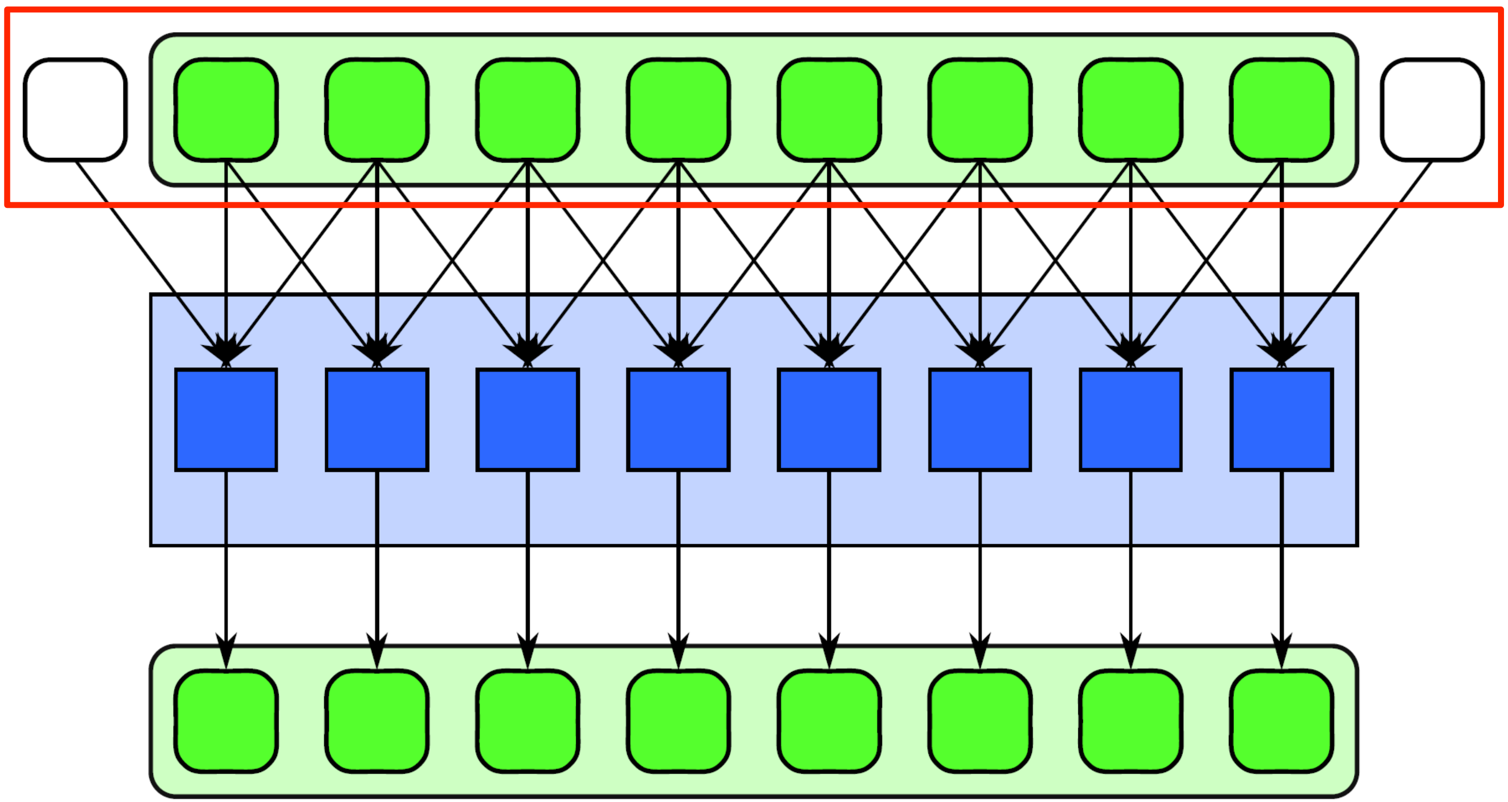
- A stencil output is a function of a "neighborhood" of elements in an input collection

What is the stencil pattern?



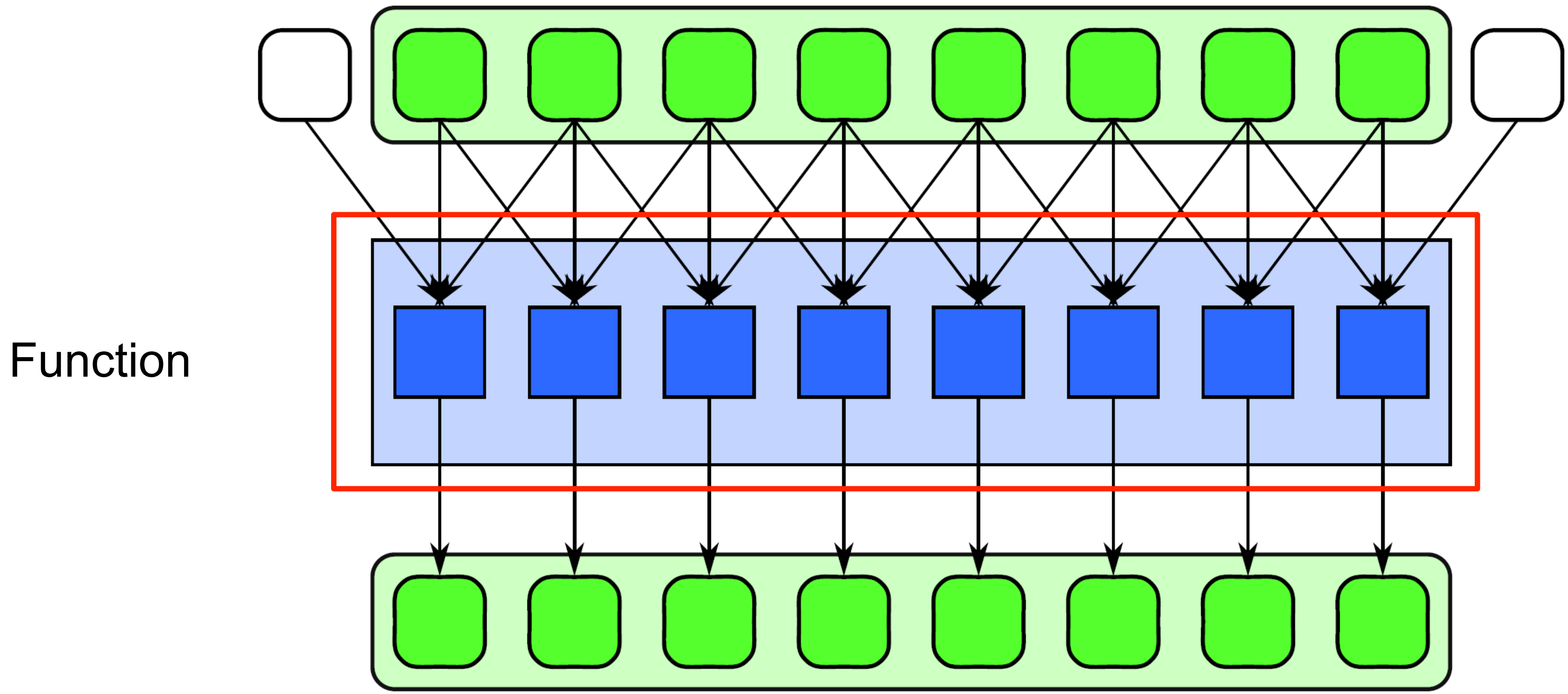
What is the stencil pattern?

Input array

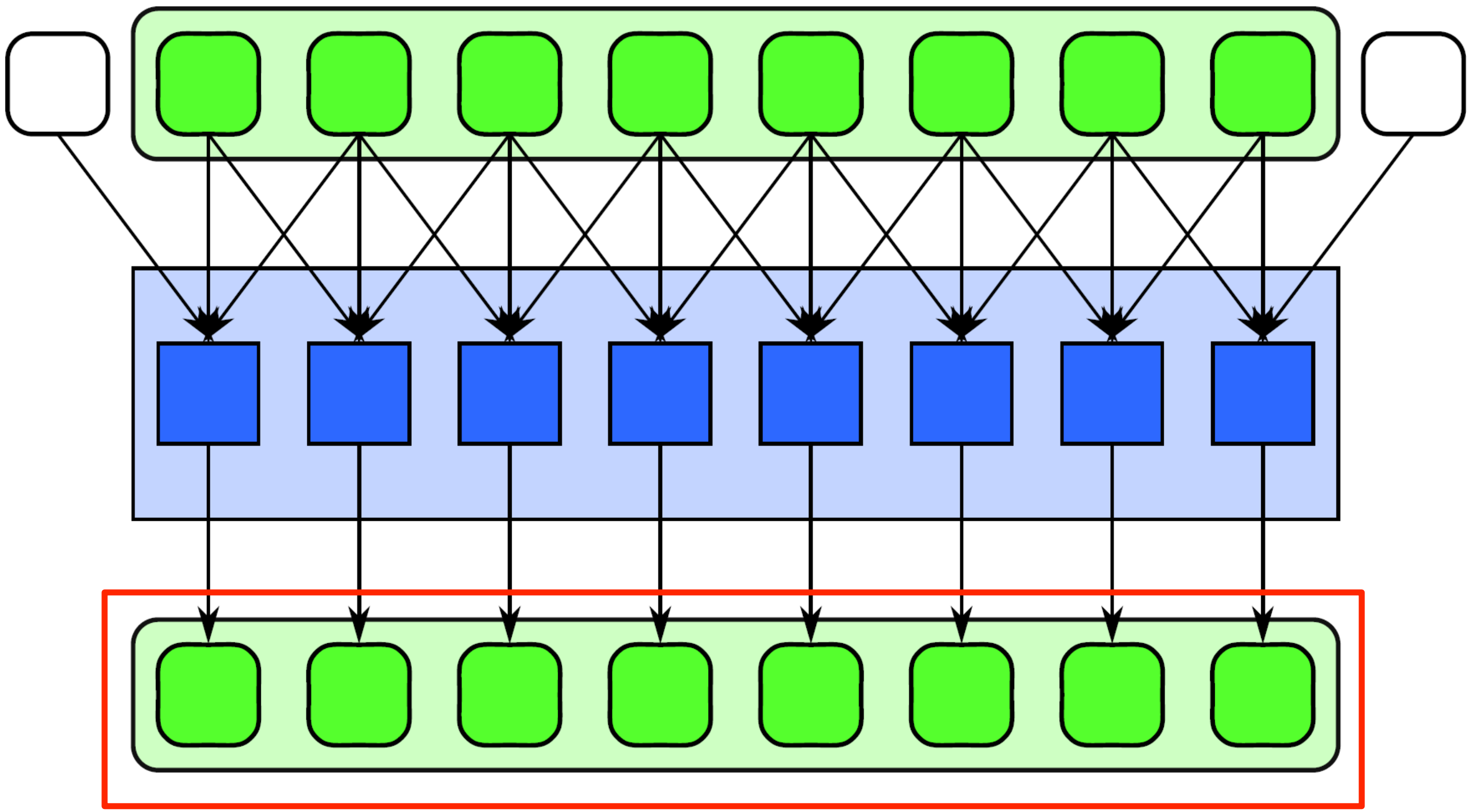


UNIVERSITY OF OREGON

What is the stencil pattern?



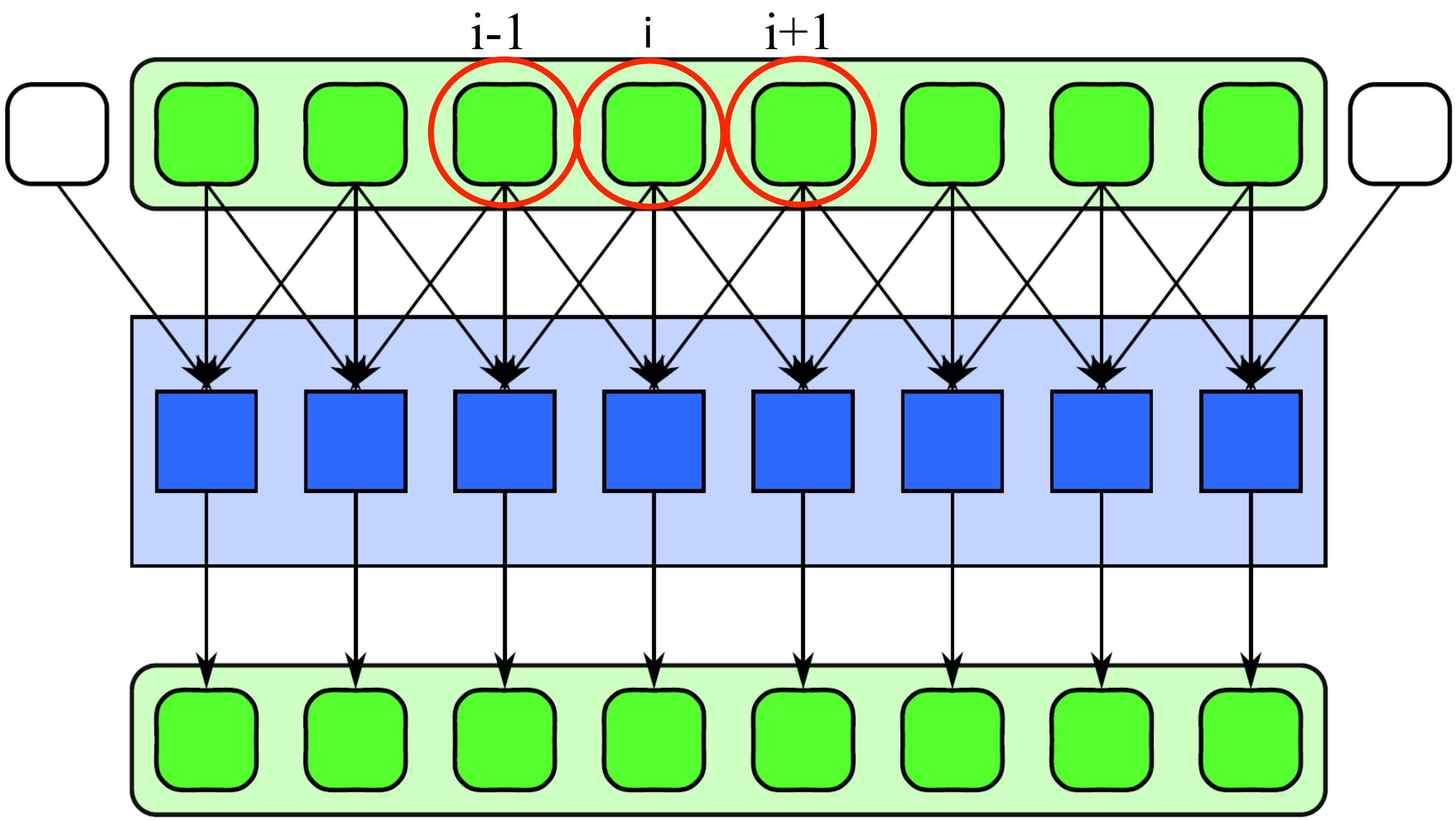
What is the stencil pattern?



Output Array

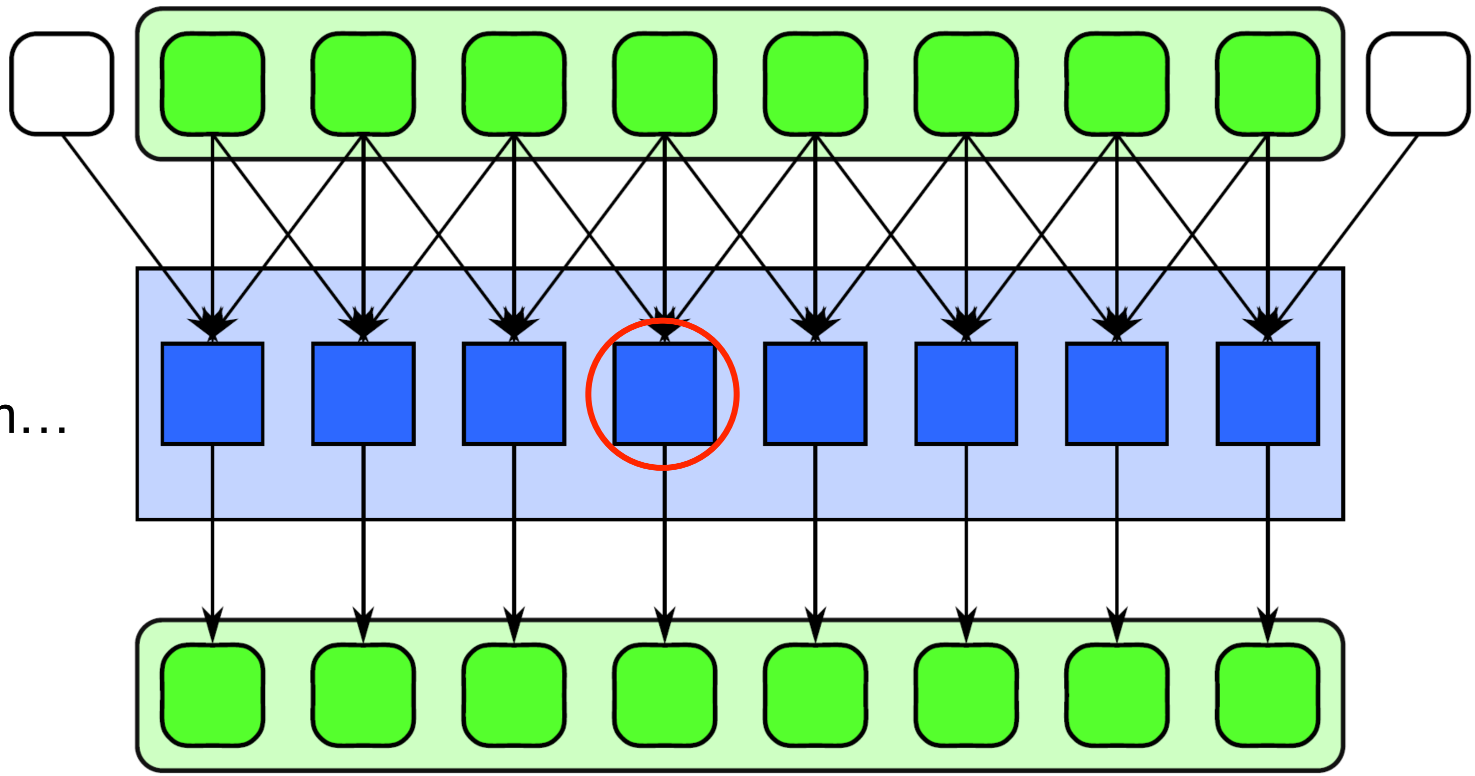
What is the stencil pattern?

This stencil takes the i-1, i, i+1 elements…

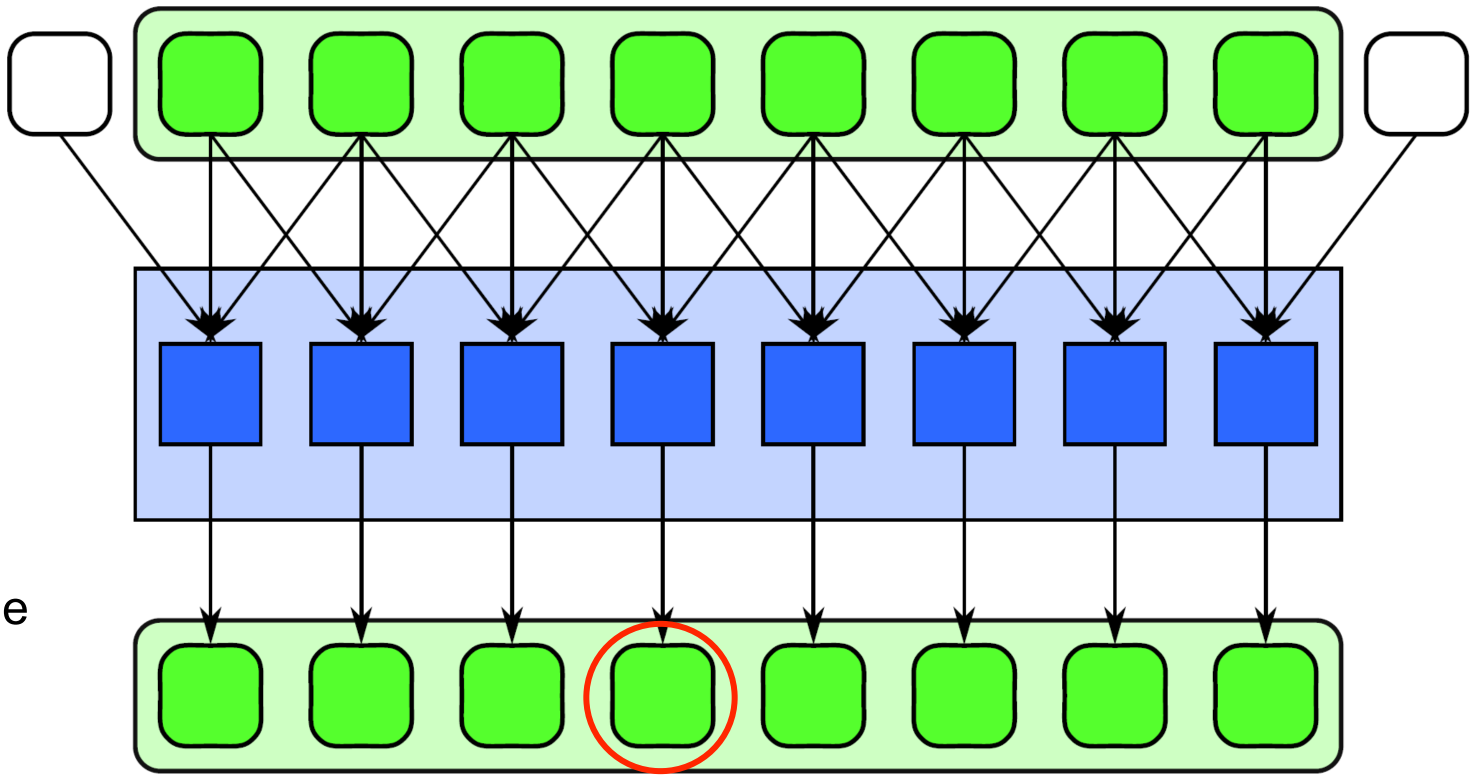


What is the stencil pattern?

Applies some function to them…



What is the stencil pattern?



And outputs to the i^{th} position of the output array

UNIVERSITY OF OREGON

What is the stencil pattern?

- Stencils can operate on one dimensional and multidimensional data
- Stencil neighborhoods can range from compact to sparse, square to cube, and anything else!

What is the stencil pattern?

Practice!

What is the stencil pattern?

- Here is our array, A

0	0	0	0
0	9	7	0
0	6	4	0
0	0	0	0

What is the stencil pattern?

- Here is our array, A

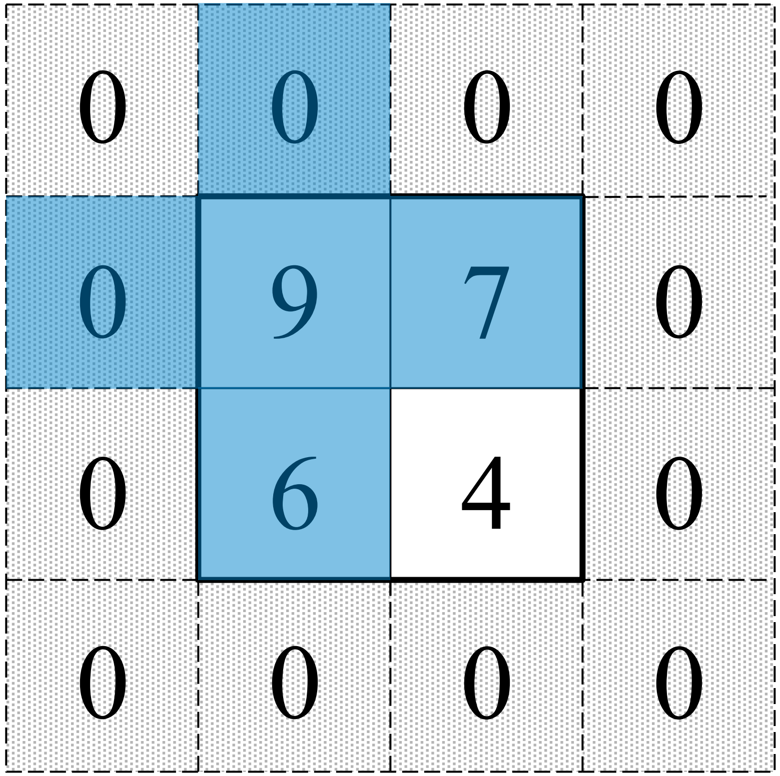
- Apply a stencil operation to the inner square of the form:

(i,j) = avg((i,j), (i-1,j), (i+1,j), (i,j-1), (i,j+1))

0	0	0	0
0	9	7	0
0	6	4	0
0	0	0	0

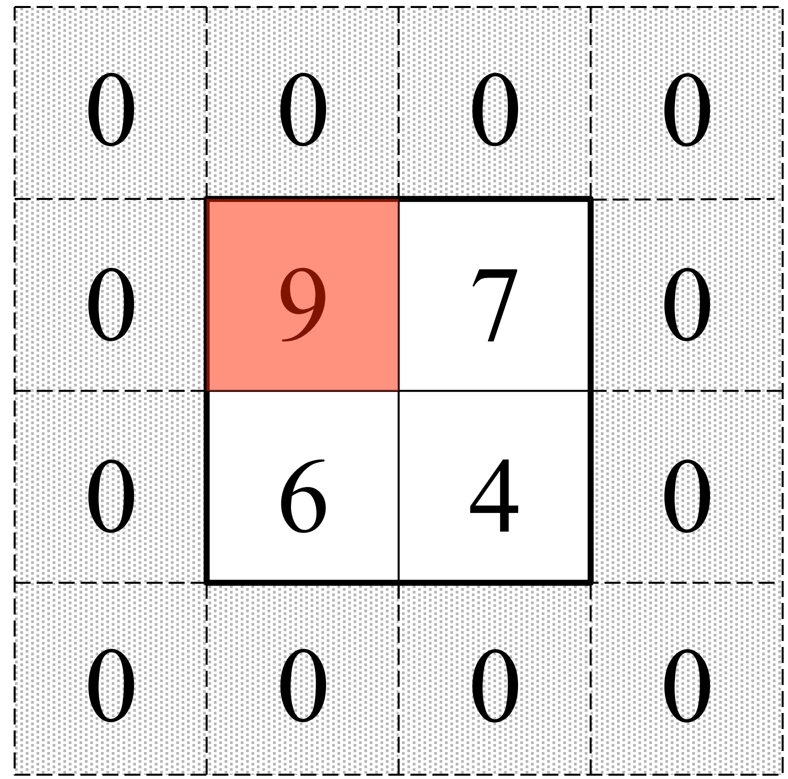
What is the stencil pattern?

1) Average all blue squares



What is the stencil pattern?

1) Average all blue squares
2) Store result in red square



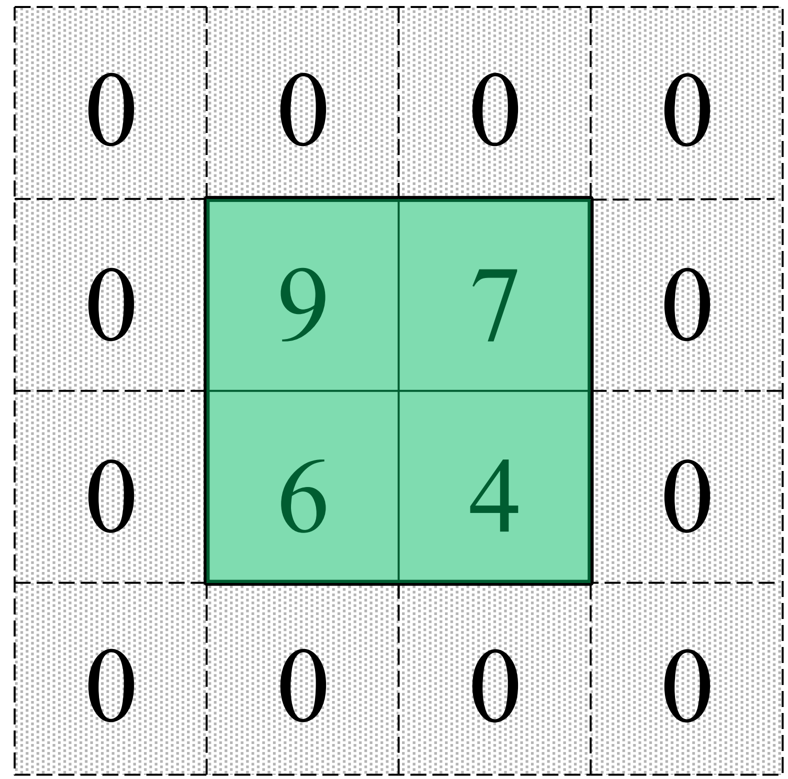
What is the stencil pattern?

1) Average all blue squares
2) Store result in red square
3) Repeat 1 and 2 for all green squares

0	0	0	0
0	9	7	0
0	6	4	0
0	0	0	0

What is the stencil pattern?

1) Average all blue squares
2) Store result in red square
3) Repeat 1 and 2 for all green squares



What is the stencil pattern?

- Output!
- How would we parallelize this?

4.4	4
3.8	3.4

UNIVERSITY OF OREGON

Serial Stencil Example (part 1)

```
template<
    int NumOff,     // number of offsets
    typename In,    // type of input locations
    typename Out,   // type of output locations
    typename F      // type of function/functor
>
void stencil(
    int n,          // number of elements in data collection
    const In a[],   // input data collection (n elements)
    Out r[],        // output data collection (n elements)
    In b,           // boundary value
    F func,         // function/functor from neighborhood inputs to output
    const int offsets[] // offsets (NumOffsets elements)
) {
```

Serial Stencil Example (part 2)

```
    // array to hold neighbors
    In neighborhood[NumOff];
    // loop over all output locations
    for (int i = 0; i < n; ++i) {
        // loop over all offsets and gather neighborhood
        for (int j = 0; j < NumOff; ++j) {
            // get index of jth input location
            int k = i+offsets[j];
            if (0 <= k && k < n) {
                // read input location
                neighborhood[j] = a[k];
            } else {
                // handle boundary case
                neighborhood[j] = b;
            }
        }
        // compute output value from input neighborhood
        r[i] = func(neighborhood);
    }
}
```

Parallel Stencil Example

Table of Contents

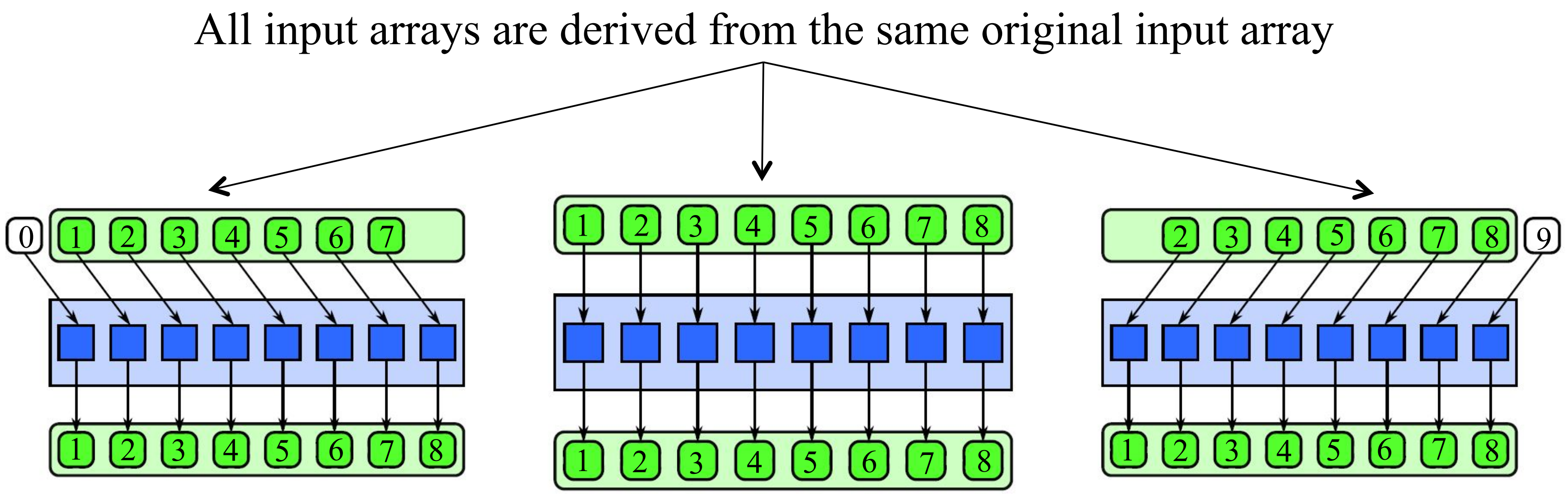
- What is the stencil pattern?
- Implementing stencil with shift
- Stencil and cache optimizations
- Stencil and communication optimizations
- Recurrence

Implementing Stencil with Shift

- One possible implementation of the stencil pattern includes shifting the input data

- For each offset in the stencil, we gather a new input vector by **shifting** the original input by the offset amount

Implementing Stencil with Shift



Implementing Stencil with Shift

- This implementation is only beneficial for one dimensional stencils or the memory-contiguous dimension of a multidimensional stencil

- Memory traffic to external memory is not reduced with shifts

- But, shifts allow vectorization of the data reads, which may reduce the total number of instructions

Table of Contents

- What is the stencil pattern?
- Implementing stencil with shift
- Stencil and cache optimizations
- Stencil and communication optimizations
- Recurrence

Stencil and Cache Optimizations

- Assuming 2D array where rows are contiguous in memory…
  - Horizontally related data will tend to belong to the same cache line
  - Vertical offset accesses will most likely result in cache misses

Stencil and Cache Optimizations

- Assigning rows to cores:
  - Maximizes horizontal data locality
  - Assuming vertical offsets in stencil, this will create redundant reads of adjacent rows from each core
- Assigning columns to cores:
  - Redundantly read data from same cache line
  - Create false sharing as cores write to same cache line

Stencil and Cache Optimizations

- Assigning "strips" to each core can be a better solution
- **Strip-mining**: an optimization in a stencil computation that groups elements in a way that avoids redundant memory accesses and aligns memory accesses with cache lines

Stencil and Cache Optimizations

- A strip's size is a multiple of a cache line in width, and the height of the 2D array
- Strip widths are in increments of the cache line size so as to avoid false sharing and redundant reads
- Each strip is processed serially from top to bottom within each core

Stencil and Cache Optimizations

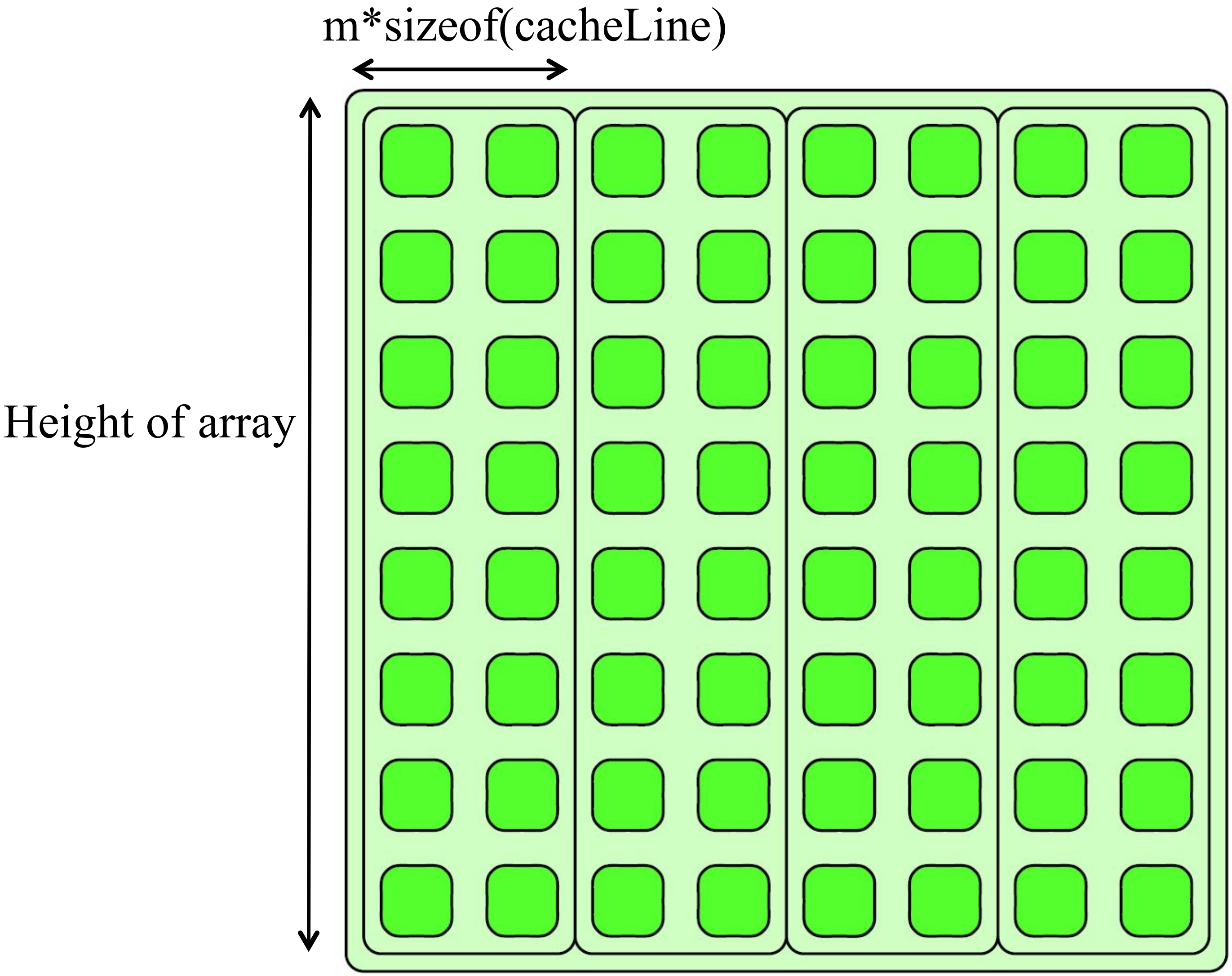
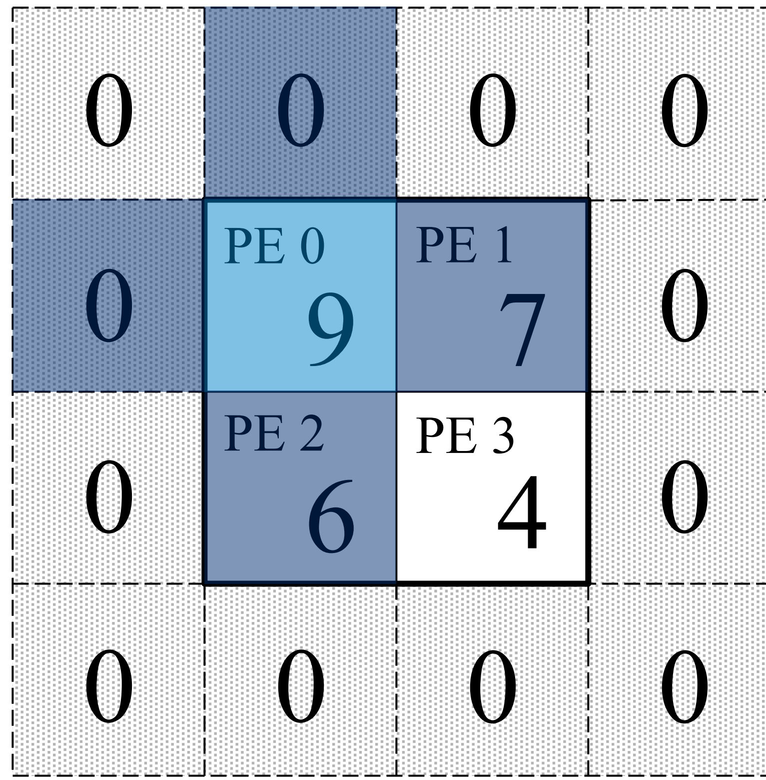


Table of Contents

Stencil and Communication Optimizations

- When data is distributed, overlapping regions (or "**ghost cells**") must be explicitly communicated between nodes between loop iterations

- Darker cells are PE 0's ghost cells
- After first iteration of stencil computation
  - PE 1 & PE 2 must send their stencil results to PE 0
  - PE 0 can perform another iteration of stencil



Stencil and Communication Optimizations

- Generally better to replicate ghost cells in each local memory and swap after each iteration than to share memory
  - Fine-grained sharing can lead to increased communication cost

Stencil and Communication Optimizations

- **Halo**: set of all ghost cells
- Halo must contain all neighbors needed for one iteration
- Larger halo (**deep halo**)
  - Less communications
  - More redundant local computation

- **Latency Hiding**: Compute interior of stencil while waiting for ghost cell updates

Table of Contents

- What is the stencil pattern?
- Implementing stencil with shift
- Stencil and cache optimizations
- Stencil and communication optimizations
- Recurrence

Recurrence

- What if we have several nested loops with data dependencies between them when doing a stencil computation?

Recurrence

```
void my_recurrence(
    size_t v,           // number of elements vertically
    size_t h,           // number of elements horizontally
    const float a[v][h], // input 2D array
    float b[v][h]       // output 2D array (boundaries already initialized )
) {
    for (int i=1; i<v; ++i)
        for (int j=1; j<h; ++j)
            b[i][j] = f(b[i-1][j], b[i][j-1], a[i][j]);
}
```

Recurrence

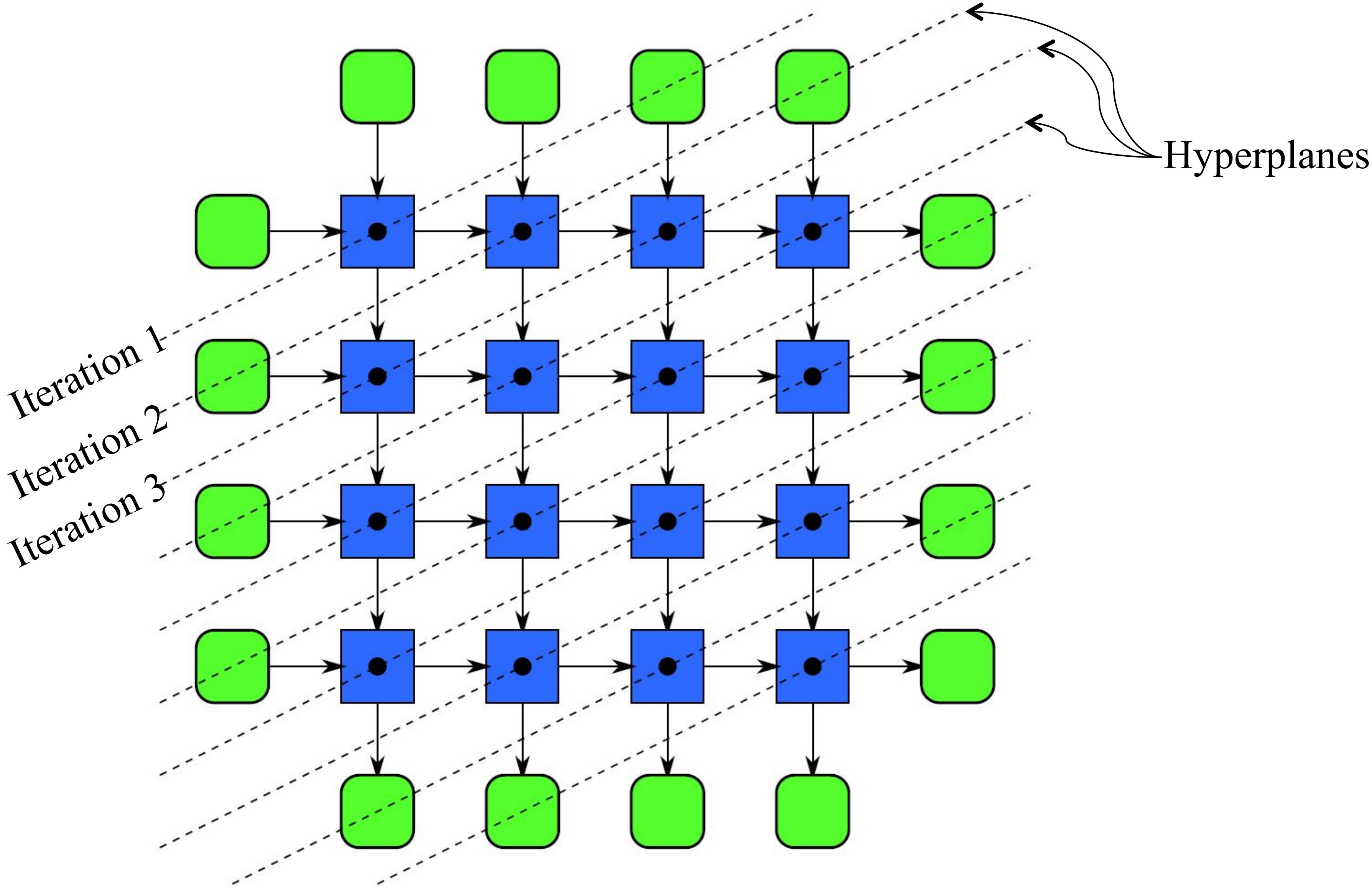
```
void my_recurrence(
    size_t v,          // number of elements vertically
    size_t h,          // number of elements horizontally
    const float a[v][h], // input 2D array
    float b[v][h]      // output 2D array (boundaries already initialized )
) {
    for (int i=1; i<v; ++i)
        for (int j=1; j<h; ++j)
            b[i][j] = f(b[i-1][j], b[i][j-1], a[i][j]);
}
```

Data dependencies between loops

Recurrence

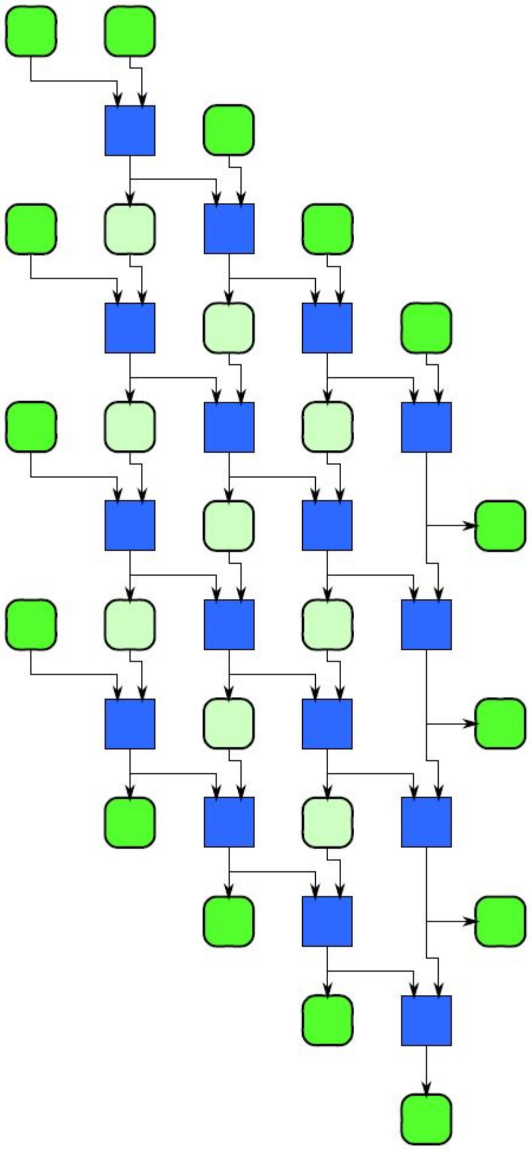
- This can still be parallelized!
- Trick: find a plane that cuts through grid of intermediate results
  - Previously computed values on one side of plane
  - Values to still be computed on other side of plane
  - Computation proceeds perpendicular to plane through time (this is known as a sweep)
- This plane is called a *separating hyperplane*

Recurrence



Recurrence

- Same grid of intermediate results
- Each level corresponds to a loop iteration
- Computation proceeds downward



Example Implementation

Conclusion

- Examined the stencil and recurrence pattern
  - Both have a regular pattern of communication and data access
- In both patterns we can convert a set of offset memory accesses to shifts
- Stencils can use strip-mining to optimize cache use
- Ghost cells should be considered when stencil data is distributed across different memory spaces