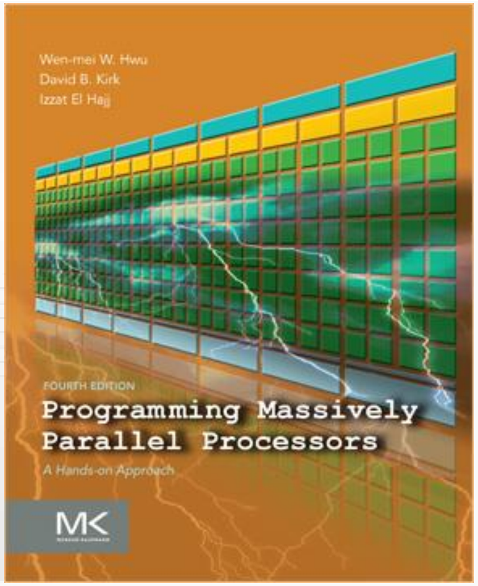
Programming Massively Parallel Processors

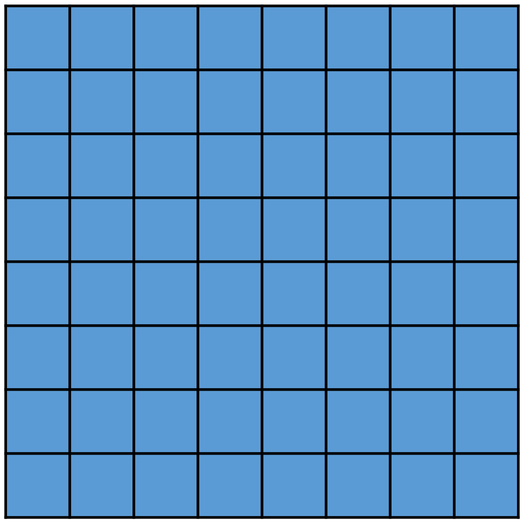
A Hands-on Approach

CHAPTER 14 (PART 1) Sparse Matrix Computation

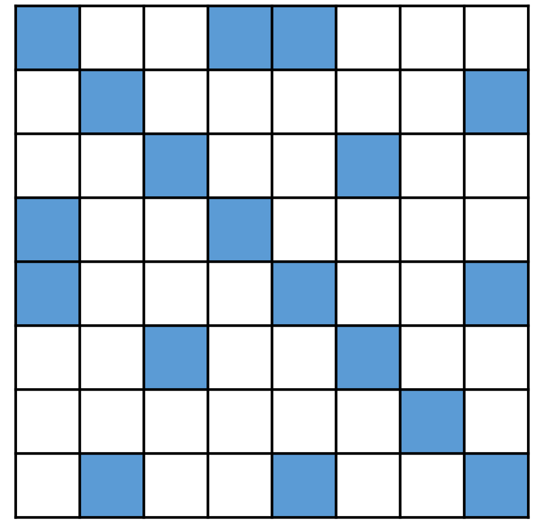


Sparse Matrices

A **dense matrix** is one where the majority of elements are not zero



A **sparse matrix** is one where many elements are zero
(many real world systems are sparse)



- Opportunities:
  - Do not need to allocate space for zeros (save memory capacity)
  - Do not need to load zeros (save memory bandwidth)
  - Do not need to compute with zeros (save computation time)

Sparse Matrix Storage Formats

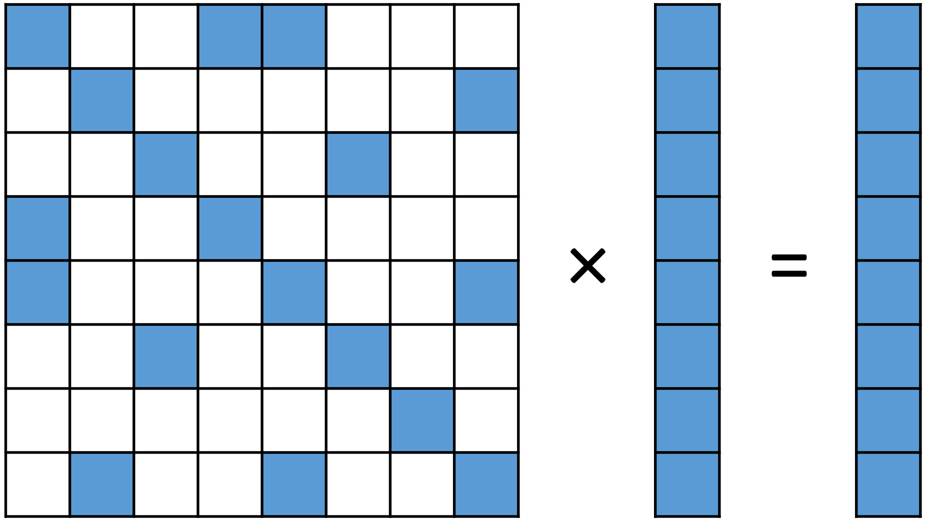
- Many storage formats for sparse matrices:
  - Coordinate Format (COO)
  - Compressed Sparse Row (CSR)
  - ELLPACK Format (ELL)
  - Jagged Diagonal Storage (JDS)
  - …

- Format design considerations:
  - Space efficiency (memory consumed)
  - Flexibility (ease of adding/reordering elements)
  - Accessibility (ease of finding desired data)
  - Memory access pattern (enabling coalescing)
  - Load balance (minimizing control divergence)

SpMV

- Choice of best format depends on computation

- We will use **Sparse Matrix-Vector multiplication** (SpMV) as an example to study different formats



Coordinate Format (COO)

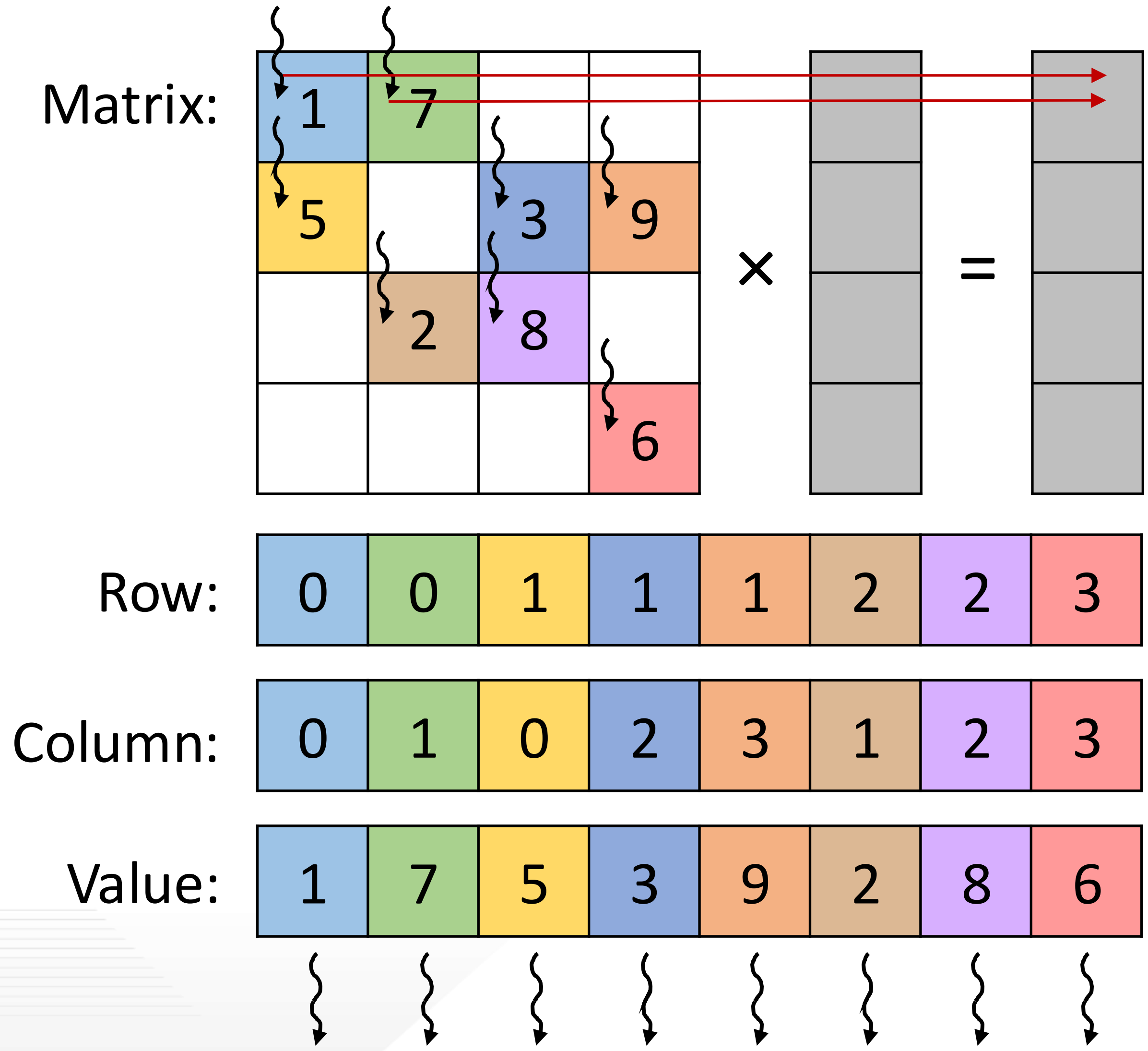
Matrix:

1	7		
5		3	9
	2	8	
			6

Store every nonzero along with its row index and column index

Row:	0	0	1	1	1	2	2	3
Column:	0	1	0	2	3	1	2	3
Value:	1	7	5	3	9	2	8	6

SpMV/COO



Multiple threads writing to the same output (need atomic operations)

Parallelization approach:

Assign one thread per nonzero

SpMV/COO Code

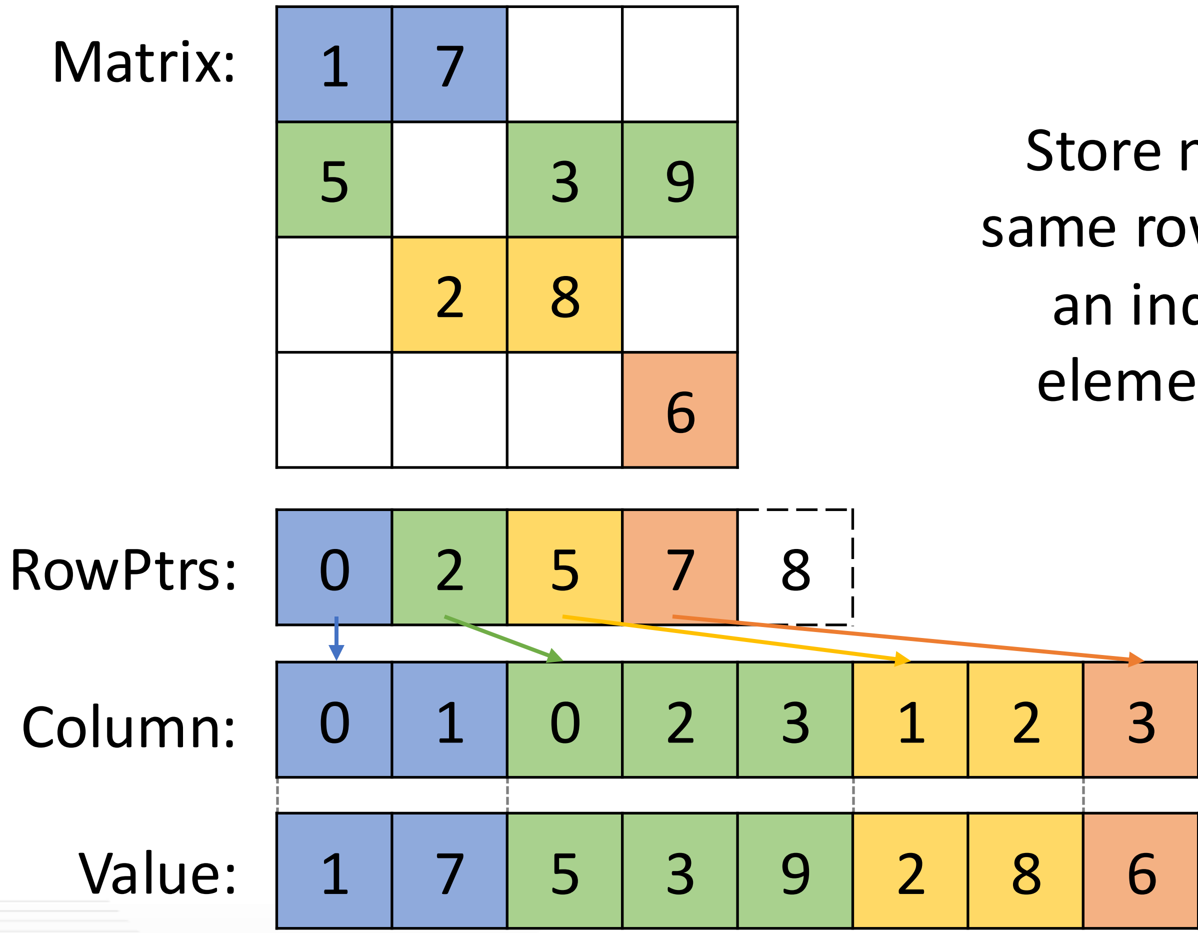
```
__global__ void spmv_coo_kernel(COOMatrix cooMatrix, float* inVector, float* outVector) {
    unsigned int i = blockIdx.x*blockDim.x + threadIdx.x;
    if(i < cooMatrix.numNonzeros) {
        unsigned int row = cooMatrix.rowIdxs[i];
        unsigned int col = cooMatrix.colIdxs[i];
        float value = cooMatrix.values[i];
        atomicAdd(&outVector[row], inVector[col]*value);
    }
}
```

COO Tradeoffs

- Advantages:
  - Flexibility: easy to add new elements to the matrix, nonzeros can be stored in any order
  - Accessibility: given nonzero, easy to find row and column
  - SpMV/COO has coalesced memory accesses
  - SpMV/COO has no control divergence

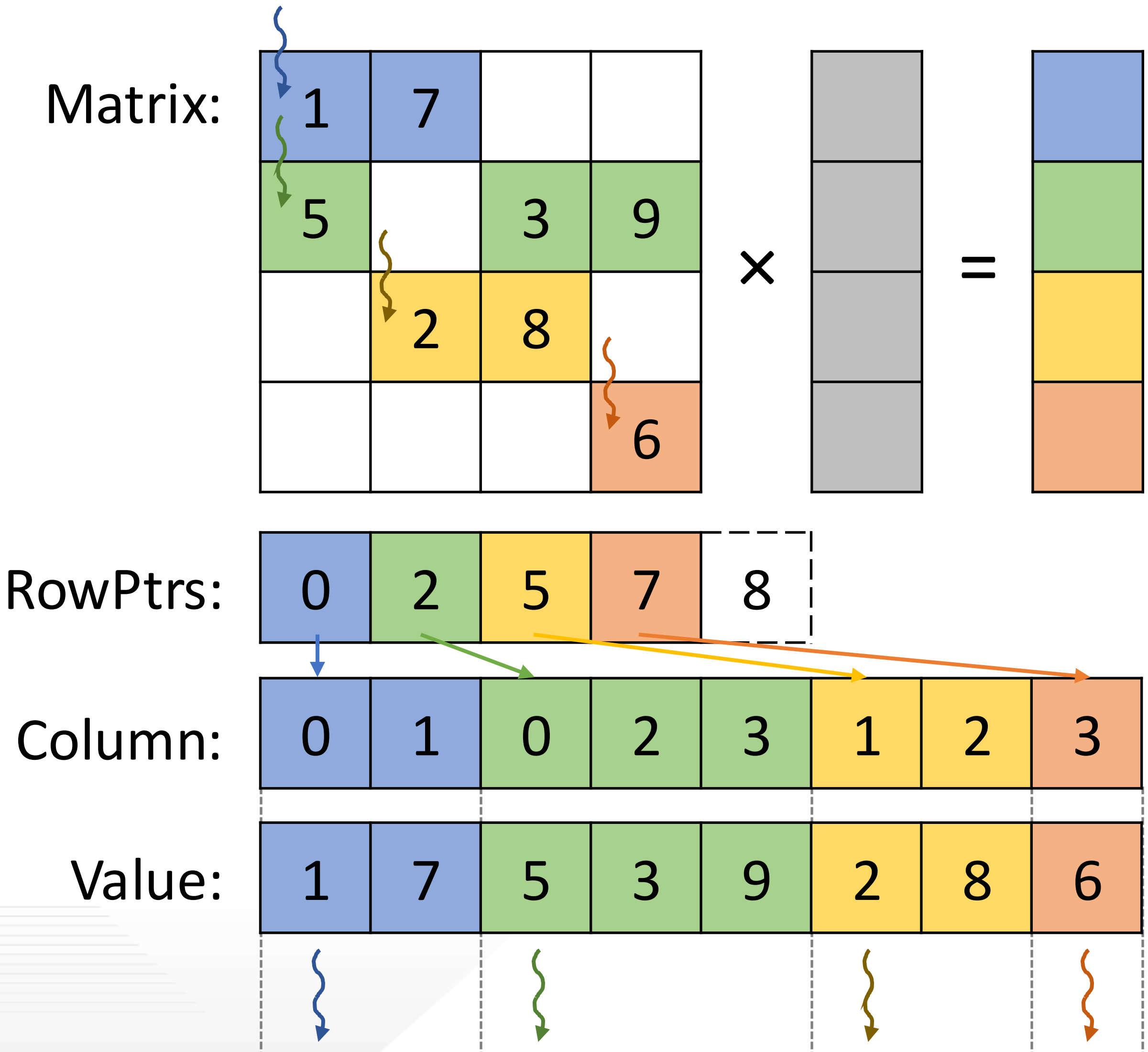
- Disadvantage:
  - Accessibility: given a row or column, hard to find all nonzeros (need to search)
  - SpMV/COO uses atomic operations

Compressed Sparse Row (CSR)



Store nonzeros of the same row adjacently and an index to the first element of each row

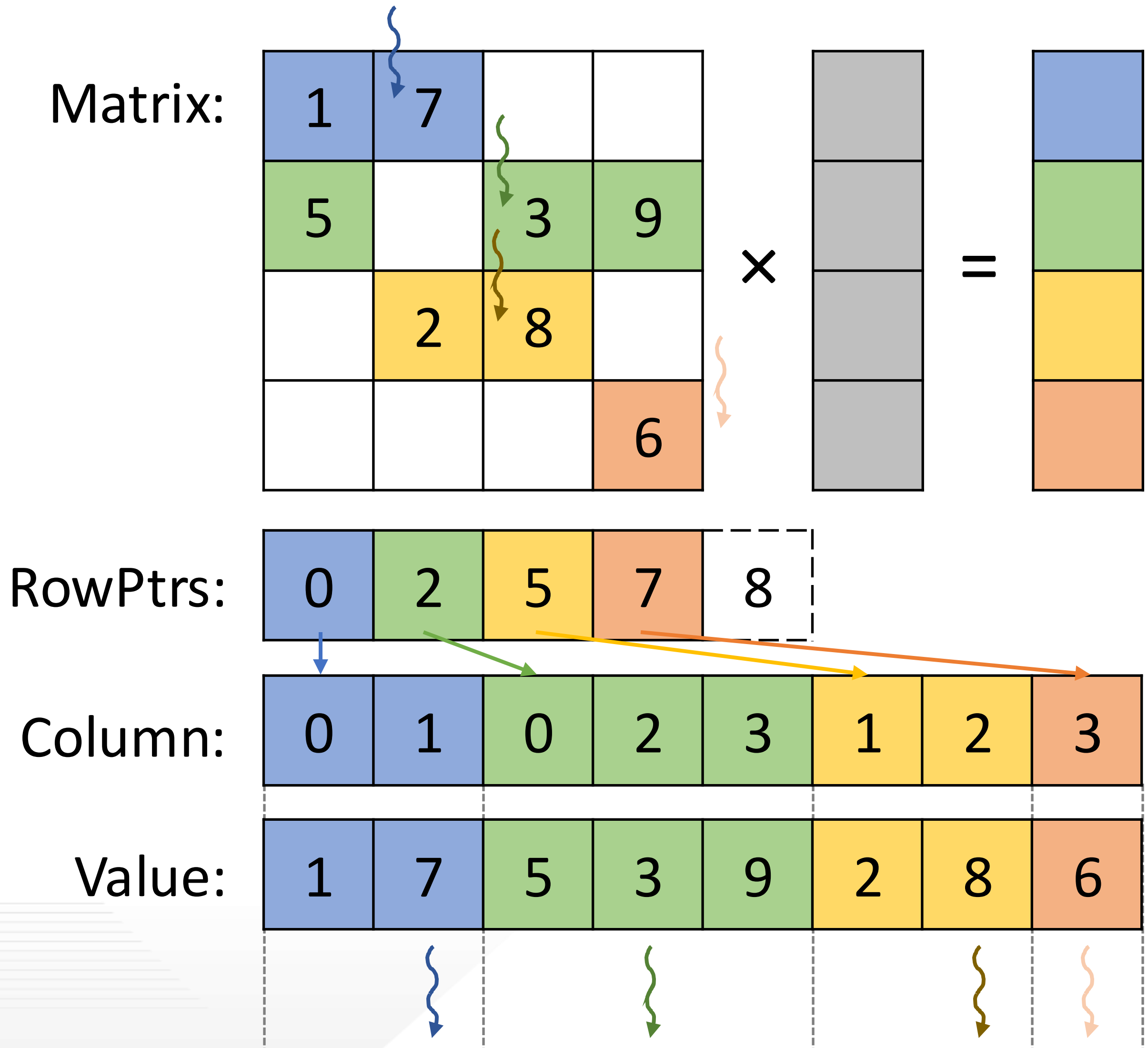
SpMV/CSR



Parallelization approach:

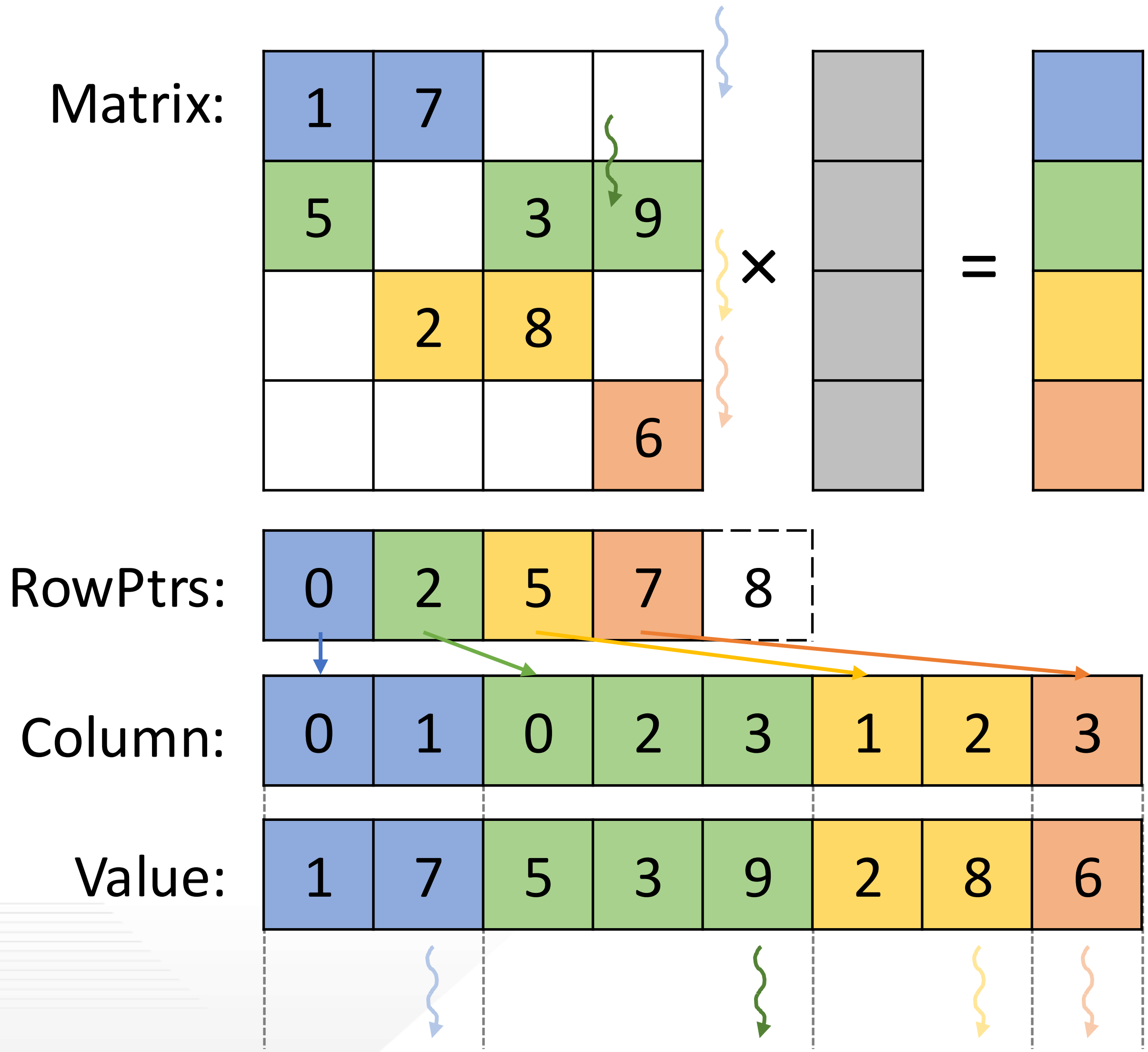
Assign one thread to loop over each input row sequentially and update corresponding output element

SpMV/CSR



Parallelization approach:
Assign one thread to loop over each input row sequentially and update corresponding output element

SpMV/CSR



Parallelization approach:
Assign one thread to loop over each input row sequentially and update corresponding output element

SpMV/CSR Code

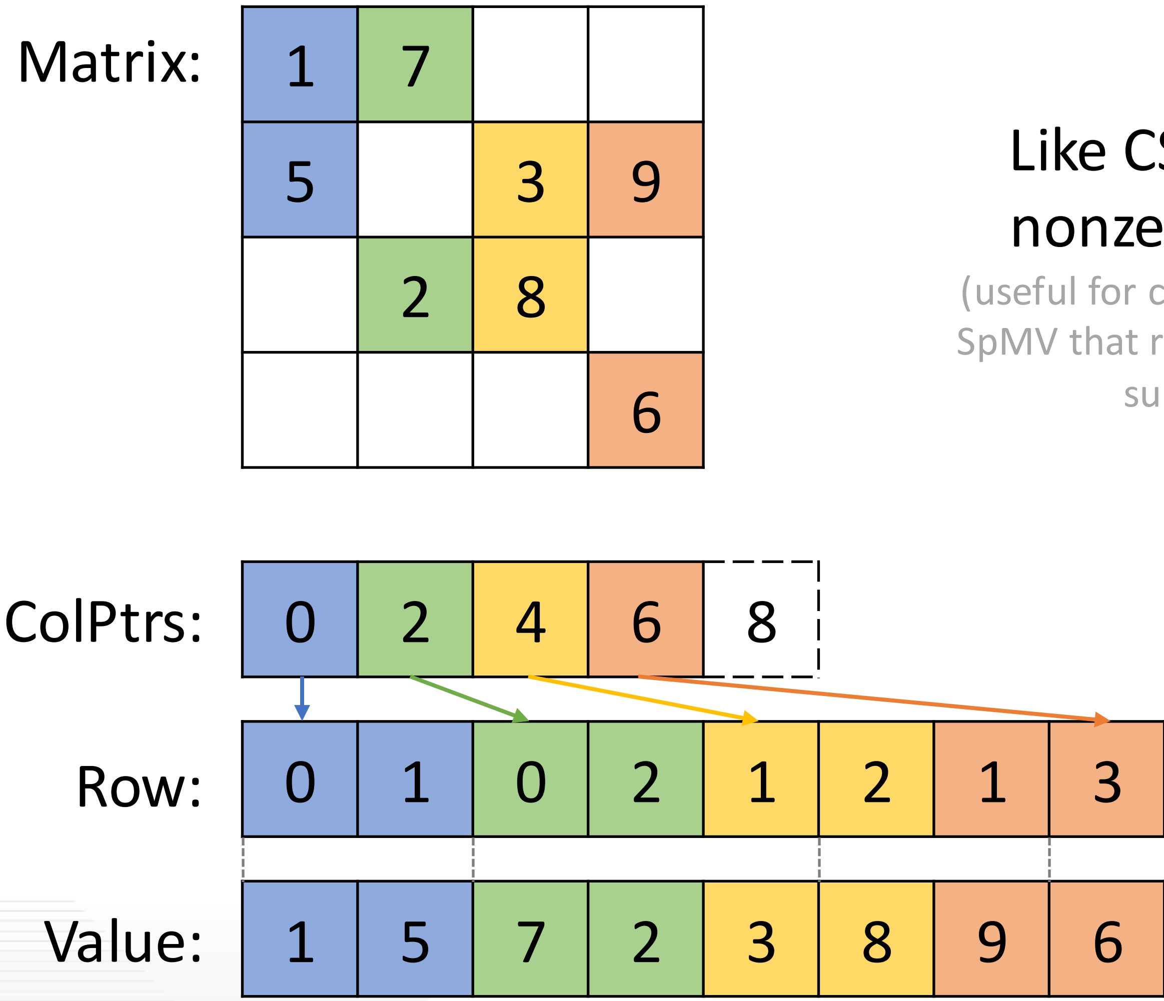
```
__global__ void spmv_csr_kernel(CSRMatrix csrMatrix, float* inVector, float* outVector) {
    unsigned int row = blockIdx.x*blockDim.x + threadIdx.x;
    if(row < csrMatrix.numRows) {
        float sum = 0.0f;
        for(unsigned int i = csrMatrix.rowPtrs[row]; i < csrMatrix.rowPtrs[row + 1]; ++i) {
            unsigned int col = csrMatrix.colIdxs[i];
            float value = csrMatrix.values[i];
            sum += inVector[col]*value;
        }
        outVector[row] = sum;
    }
}
```

CSR Tradeoffs (versus COO)

- Advantages:
  - Space efficiency: row pointers smaller than row indexes
  - Accessibility: given a row, easy to find all nonzeros
  - SpMV/CSR avoids atomics, every thread owns its output

- Disadvantage:
  - Flexibility: hard to add new elements to the matrix
  - Accessibility: given nonzero, hard to find row; given a column, hard to find all nonzeros
  - SpMV/CSR memory accesses are not coalesced
  - SpMV/CSR has control divergence

Compressed Sparse Column (CSC)



Like CSR, but groups nonzeros by column

(useful for computations other than SpMV that require column traversal, such as SpMSpV)

References

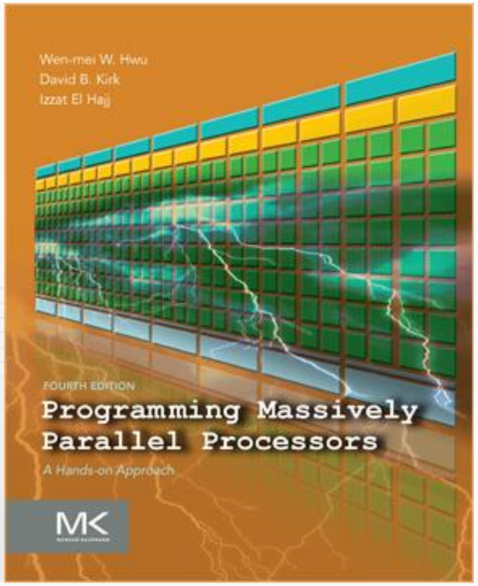
- Wen-mei W. Hwu, David B. Kirk, and Izzat El Hajj. *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann, 2022.

MORGAN KAUFMANN

Programming Massively Parallel Processors

A Hands-on Approach

CHAPTER 14 (PART 2) › Sparse Matrix Computation



Recall: CSR Tradeoffs (versus COO)

- Advantages:
  - Space efficiency: row pointers smaller than row indexes
  - Accessibility: given a row, easy to find all nonzeros
  - SpMV/CSR <u>avoids atomics</u>, every thread owns its output

- Disadvantage:
  - Flexibility: hard to add new elements to the matrix
  - Accessibility: given nonzero, hard to find row; given a column, hard to find all nonzeros
  - SpMV/CSR memory accesses are <u>not coalesced</u>
  - SpMV/CSR has <u>control divergence</u>

Can we design a format that avoids atomics but also improves memory coalescing and reduces control divergence?

ELLPACK Format (ELL)

Matrix:

1	7		
5		3	9
	2	8	
			6

Group nonzeros by row (like CSR)…

Column:

0	1	
0	2	3
1	2	
3		

Value:

1	7	
5	3	9
2	8	
6		

ELLPACK Format (ELL)

Matrix:

1	7		
5		3	9
	2	8	
			6

...pad rows so they all have the same size...

Column:

0	1	*
0	2	3
1	2	*
3	*	*

Value:

1	7	*
5	3	9
2	8	*
6	*	*

ELLPACK Format (ELL)

Matrix:

1	7		
5		3	9
	2	8	
			6

…and store padded array of nonzeros in column major order

Column:

0	1	*
0	2	3
1	2	*
3	*	*

Value:

1	7	*
5	3	9
2	8	*
6	*	*

ELLPACK Format (ELL)

Matrix:

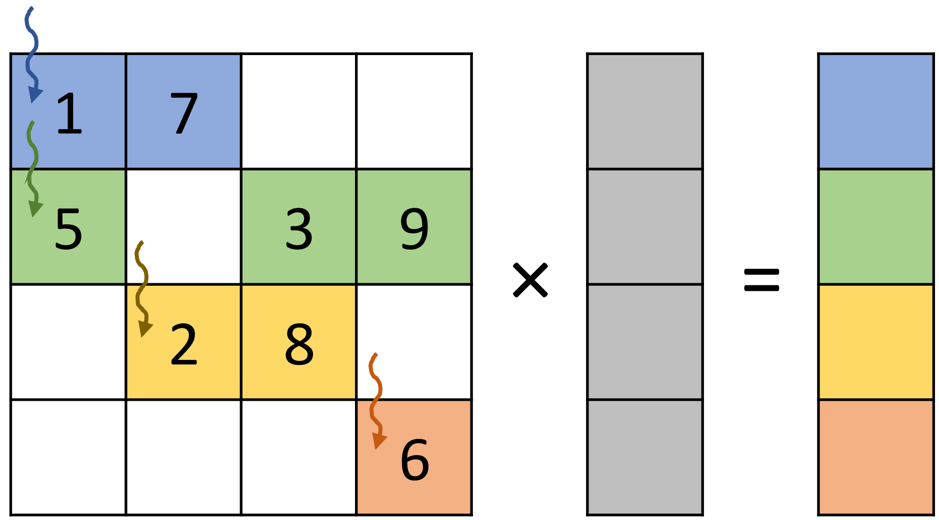
1	7		
5		3	9
	2	8	
			6

…and store padded array of nonzeros in column major order

Column:	0	0	1	3	1	2	2	*	*	3	*	*
Value:	1	5	2	6	7	3	8	*	*	9	*	*

SpMV/ELL

Matrix:



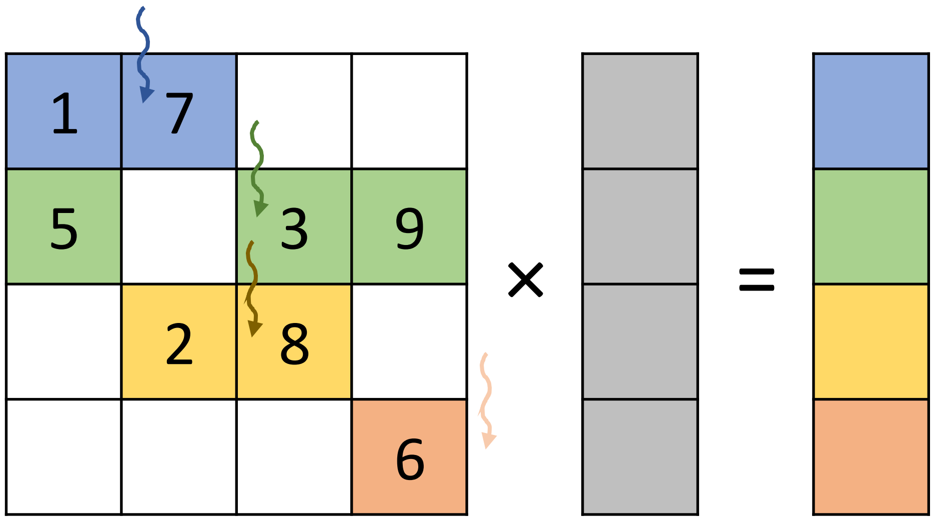
Parallelization approach:
Assign one thread to loop over each input row sequentially and update corresponding output element

Column:	0	0	1	3	1	2	2	*	*	3	*	*
Value:	1	5	2	6	7	3	8	*	*	9	*	*

Memory accesses are coalesced

SpMV/ELL

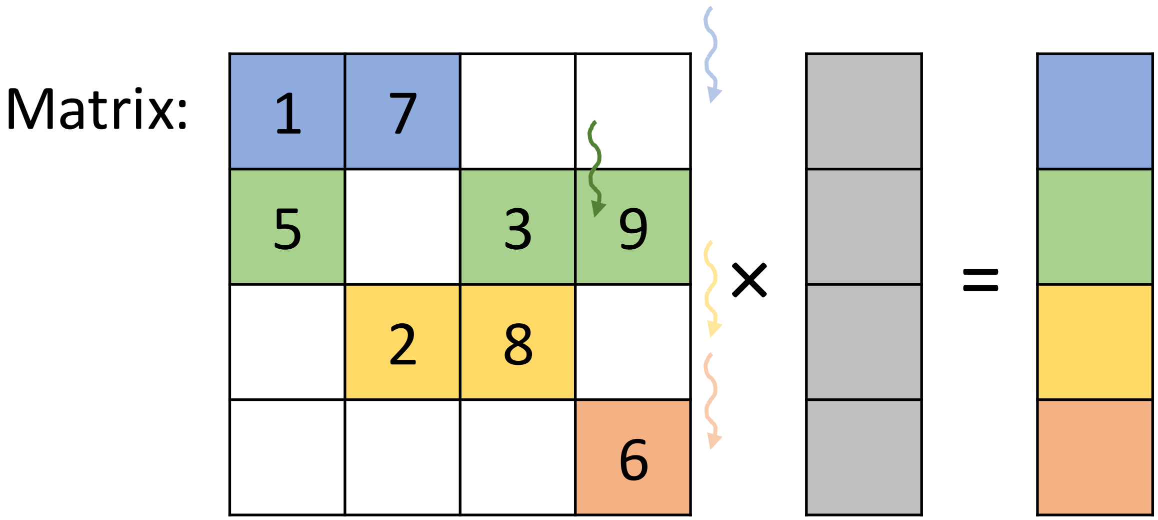
Matrix:



Parallelization approach:
Assign one thread to loop over each input row sequentially and update corresponding output element

Column:	0	0	1	3	1	2	2	*	*	3	*	*
Value:	1	5	2	6	7	3	8	*	*	9	*	*

Memory accesses are coalesced



Parallelization approach:
Assign one thread to loop over each input row sequentially and update corresponding output element

Column:	0	0	1	3	1	2	2	*	*	3	*	*
Value:	1	5	2	6	7	3	8	*	*	9	*	*

Memory accesses are coalesced

SpMV/ELL Code

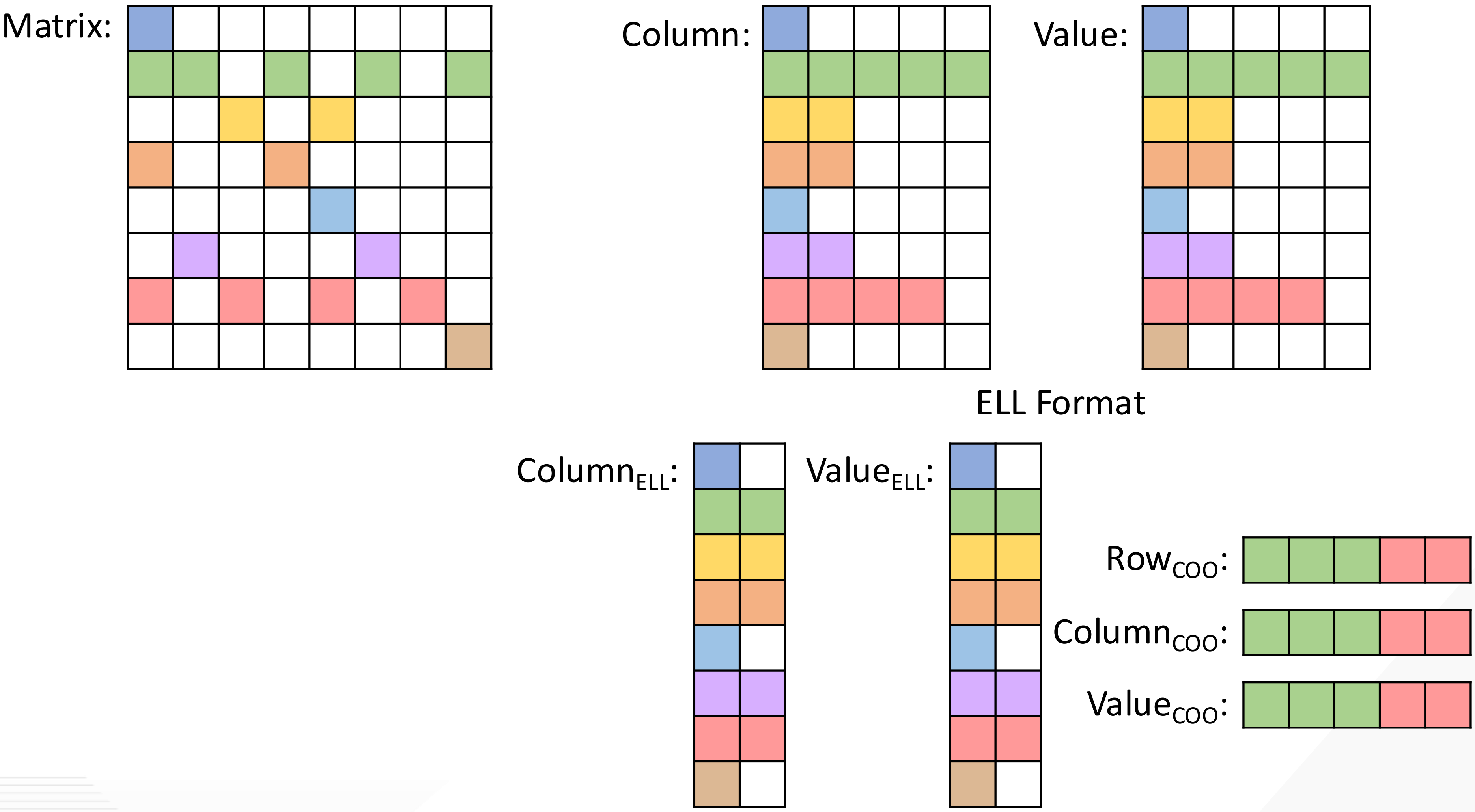
```
__global__ void spmv_ell_kernel(ELLMatrix ellMatrix, float* inVector, float* outVector) {
    unsigned int row = blockIdx.x*blockDim.x + threadIdx.x;
    if(row < ellMatrix.numRows) {
        float sum = 0.0f;
        for(unsigned int nnzIdx = 0; nnzIdx < ellMatrix.nnzPerRow[row]; ++nnzIdx) {
            unsigned int i = nnzIdx*ellMatrix.numRows + row;
            unsigned int col = ellMatrix.colIdxs[i];
            float value = ellMatrix.values[i];
            sum += inVector[col]*value;
        }
        outVector[row] = sum;
    }
}
```

ELL Tradeoffs

- Advantages:
  - Flexibility: can add new elements as long as row not full
  - Accessibility: given a row, easy to find all nonzeros; given nonzero, easy to find row and column
  - SpMV/ELL memory accesses are coalesced

- Disadvantage:
  - Space efficiency: overhead due to padding
  - Accessibility: given a column, hard to find all nonzeros
  - SpMV/ELL has control divergence

Hybrid ELL + COO



Hybrid ELL + COO

(use COO for very long rows)

ELL + COO Tradeoffs

- Similar to ELL, with the following added benefits from using COO:
  - Space efficiency: less padding
  - Flexibility: can add new elements to any row
  - Less control divergence

Jagged Diagonal Storage (JDS)

Matrix:

a					
b	c		d	e	
		f		g	
h			i		j
	k			l	
m		n			o

Group nonzeros by row (like CSR)...

Column:

0			
0	1	3	4
2	4		
0	3	5	
1	4		
0	2	5	

Value:

a			
b	c	d	e
f	g		
h	i	j	
k	l		
m	n	o	

Jagged Diagonal Storage (JDS)

Matrix:

a					
b	c		d	e	
		f		g	
h			i		j
	k			l	
m		n			o

…sort rows by size and remember the original row index…

Column:

0	1	3	4
0	3	5	
0	2	5	
2	4		
1	4		
0			

Value:

b	c	d	e
h	i	j	
m	n	o	
f	g		
k	l		
a			

Row:

1
3
5
2
4
0

Jagged Diagonal Storage (JDS)

Matrix:

a					
b	c		d	e	
		f		g	
h			i		j
	k			l	
m		n			o

…store nonzeros in column major…

Column:

0	1	3	4
0	3	5	
0	2	5	
2	4		
1	4		
0			

Value:

b	c	d	e
h	i	j	
m	n	o	
f	g		
k	l		
a			

Row:

1
3
5
2
4
0

Jagged Diagonal Storage (JDS)

Matrix:

a					
b	c		d	e	
		f		g	
h			i		j
	k			l	
m		n			o

…store nonzeros in column major…

Row:

1
3
5
2
4
0

Column:	0	0	0	2	1	0	1	3	2	4	4	3	5	5	4
Value:	b	h	m	f	k	a	c	i	n	g	l	d	j	o	e

Jagged Diagonal Storage (JDS)

Matrix:

a					
b	c		d	e	
		f		g	
h			i		j
	k			l	
m		n			o

…and remember where the nonzeros of each iteration start

IterPtr:

0	6	11	14	15

Column:

0	0	0	2	1	0	1	3	2	4	4	3	5	5	4

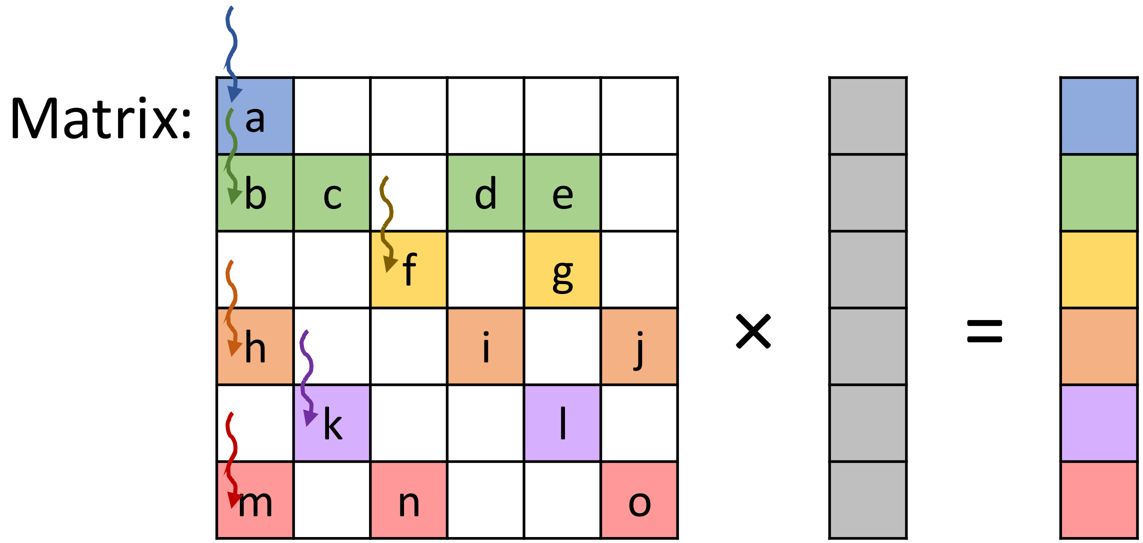
Value:

b	h	m	f	k	a	c	i	n	g	l	d	j	o	e

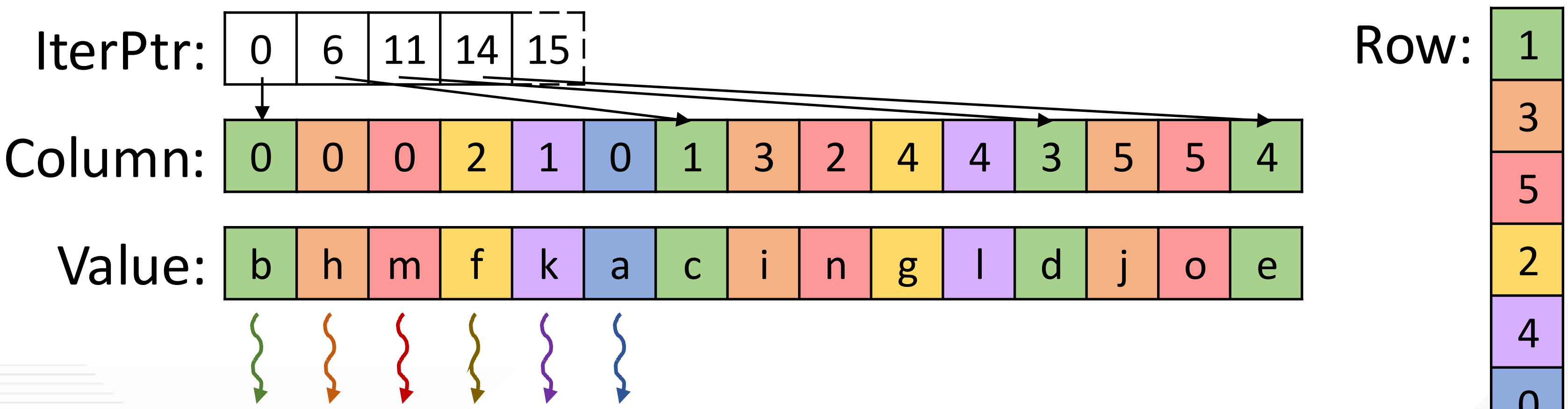
Row:

1
3
5
2
4
0

SpMV/JDS Kernel

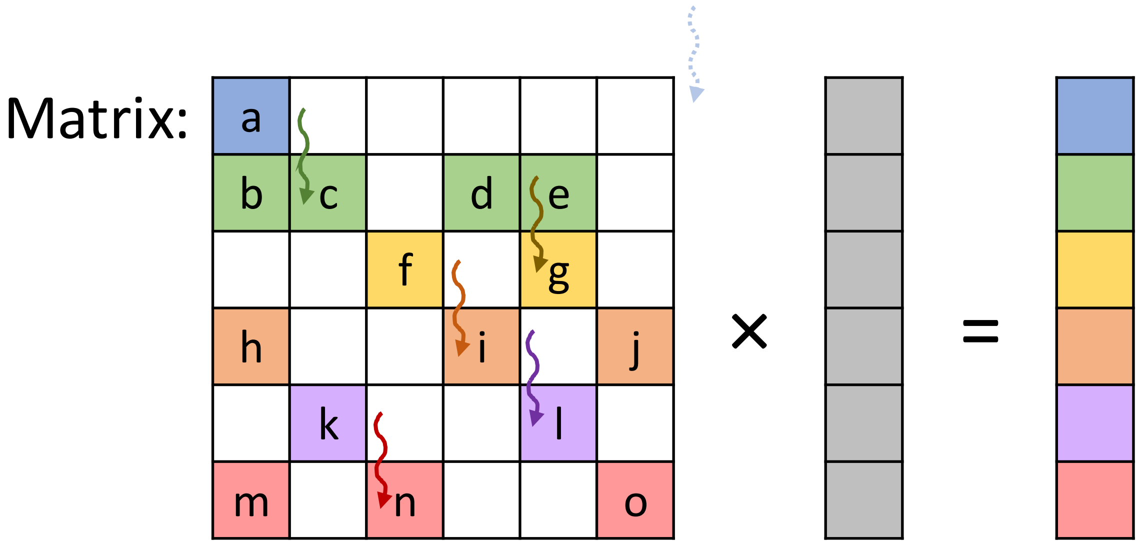


Parallelization approach:
Assign one thread to loop over each input row sequentially and update corresponding output element

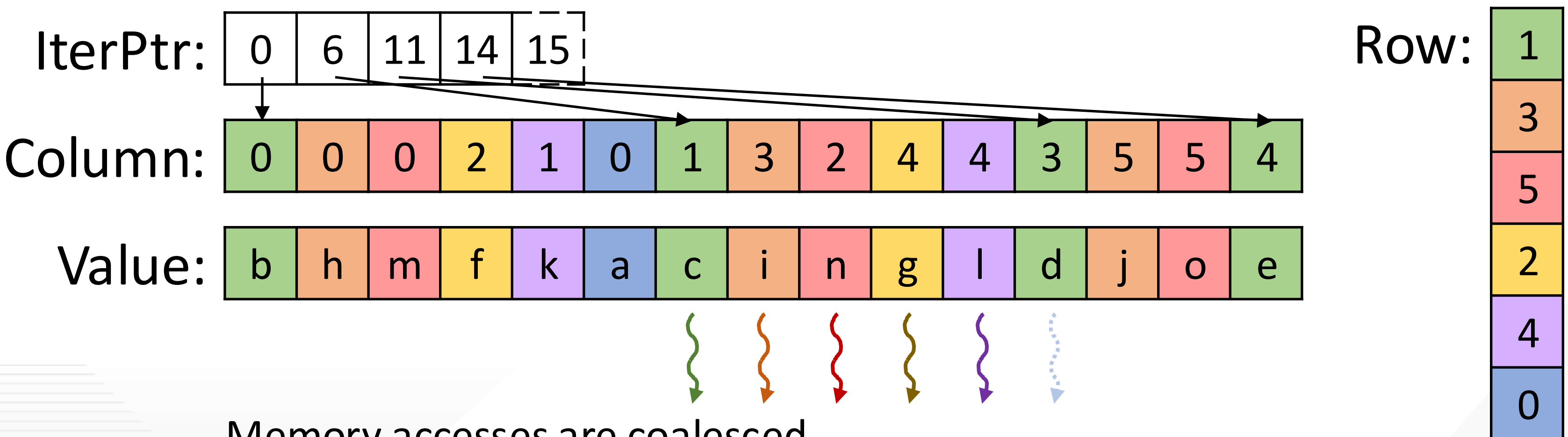


Memory accesses are coalesced…

SpMV/JDS Kernel

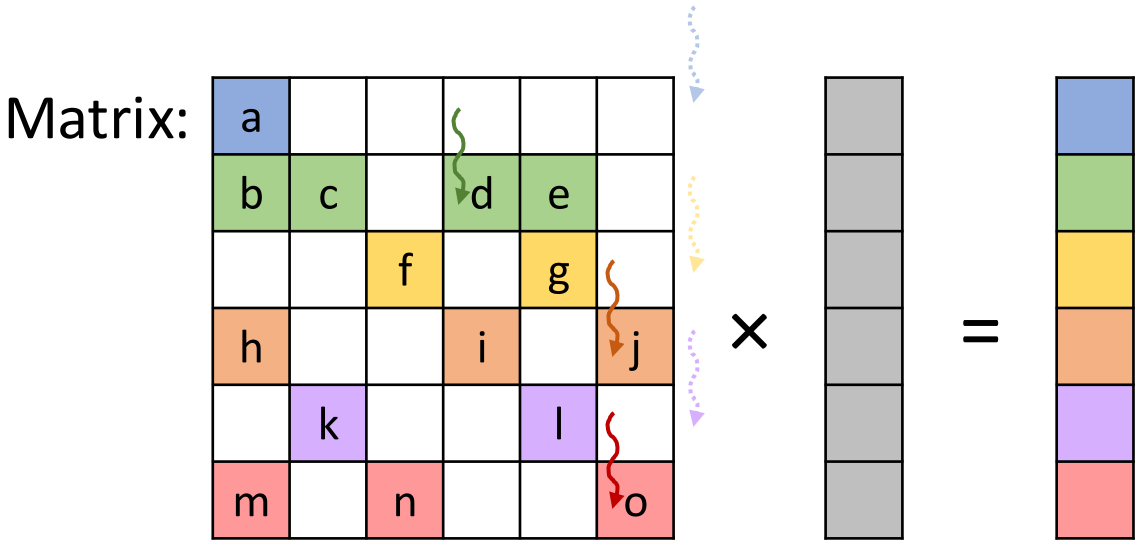


Parallelization approach:
Assign one thread to loop over each input row sequentially and update corresponding output element

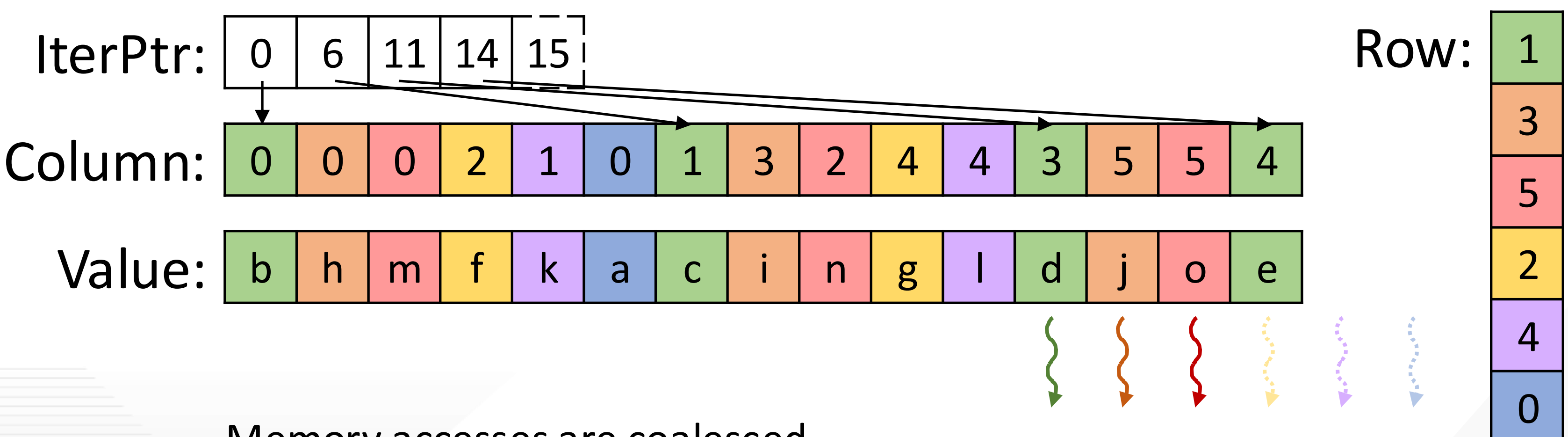


Memory accesses are coalesced…
…and threads drop out from the end, minimizing control divergence

SpMV/JDS Kernel

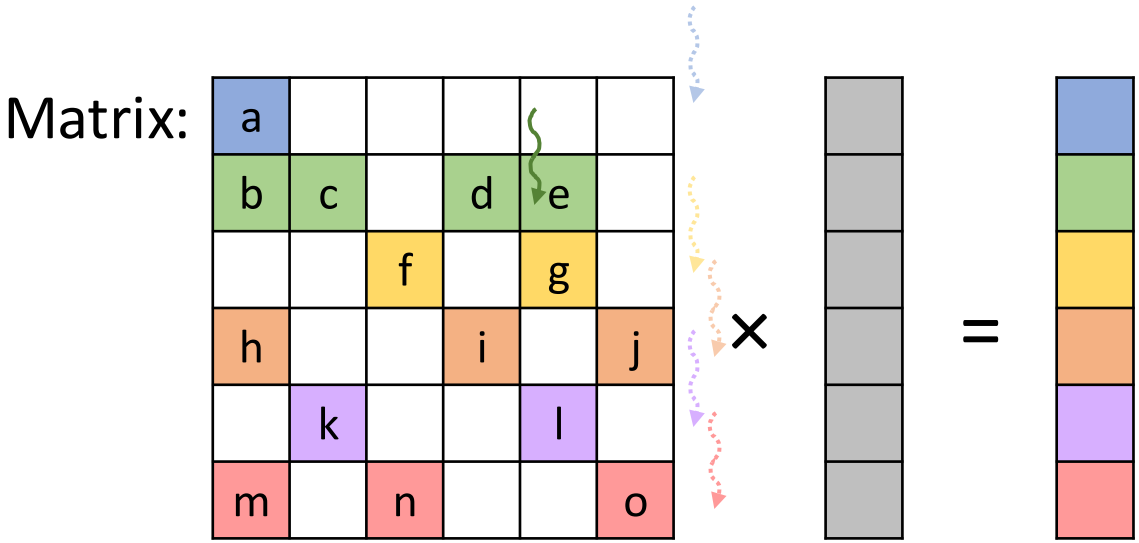


Parallelization approach:
Assign one thread to loop over each input row sequentially and update corresponding output element

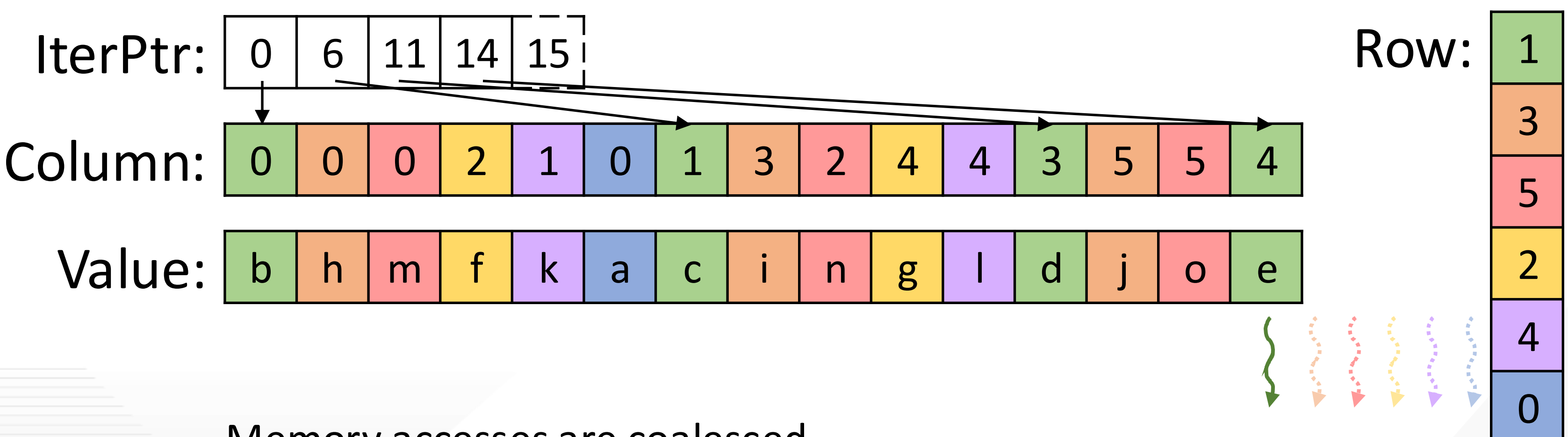


Memory accesses are coalesced…
…and threads drop out from the end, minimizing control divergence

SpMV/JDS Kernel



Parallelization approach:
Assign one thread to loop over each input row sequentially and update corresponding output element



Memory accesses are coalesced…
…and threads drop out from the end, minimizing control divergence

JDS Tradeoffs

- Advantages:
  - Space efficiency: no padding
  - Accessibility: given a row, easy to find all nonzeros
  - SpMV/JDS memory accesses are coalesced
  - SpMV/JDS minimizes control divergence

- Disadvantage:
  - Flexibility: hard to add new elements to the matrix
  - Accessibility: given nonzero, hard to find row; given a column, hard to find all nonzeros

References

- Wen-mei W. Hwu, David B. Kirk, and Izzat El Hajj. *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann, 2022.