Programming Massively Parallel Processors

Chapter14 Sparse matrix computation

242AIG18 JiYeong Yi

Chapter14

Sparse matrix computation

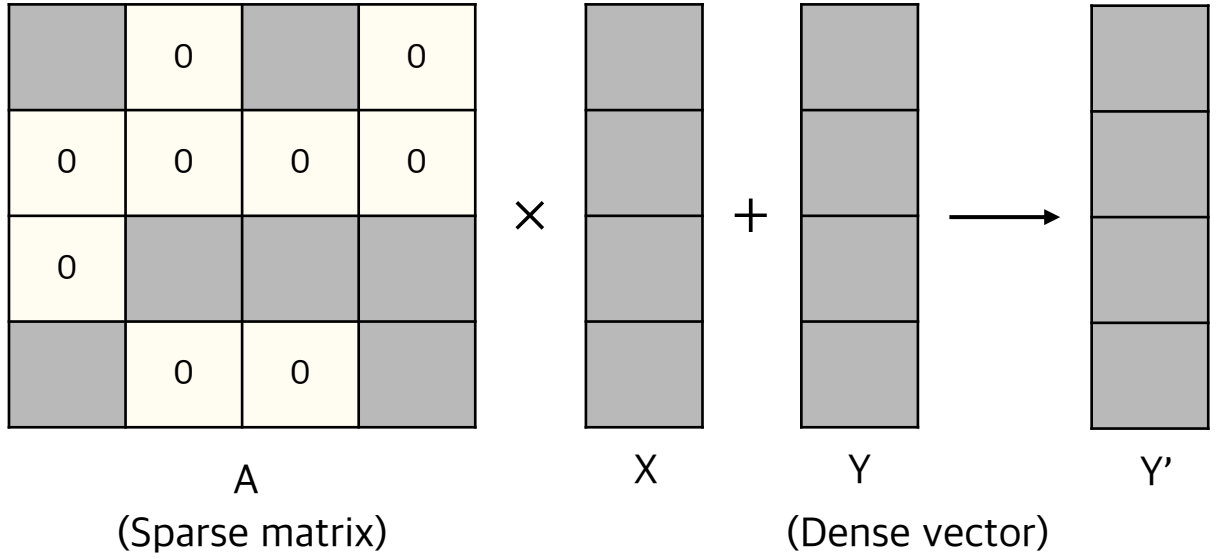
Table of contents

14.1 Background

- A sparse matrix is a matrix in which most of the elements are zeros.
  Sparse matrices are typically stored in a format, or representation, that avoids storing zero elements.

- **SpMV** (sparse matrix vector multiplication and accumulation)

$$Y' = A * X + Y$$



14.1 Background

- The main objective of the different **sparse matrix storage formats** is **to remove all the zero elements** from the matrix representation.

- Removing all the zero elements not only saves storage but also eliminates the need to fetch these zero elements from memory and perform useless multiplication or addition operations with them. This can significantly reduce the consumption of **memory bandwidth** and **computation resources**.

14.1 Background

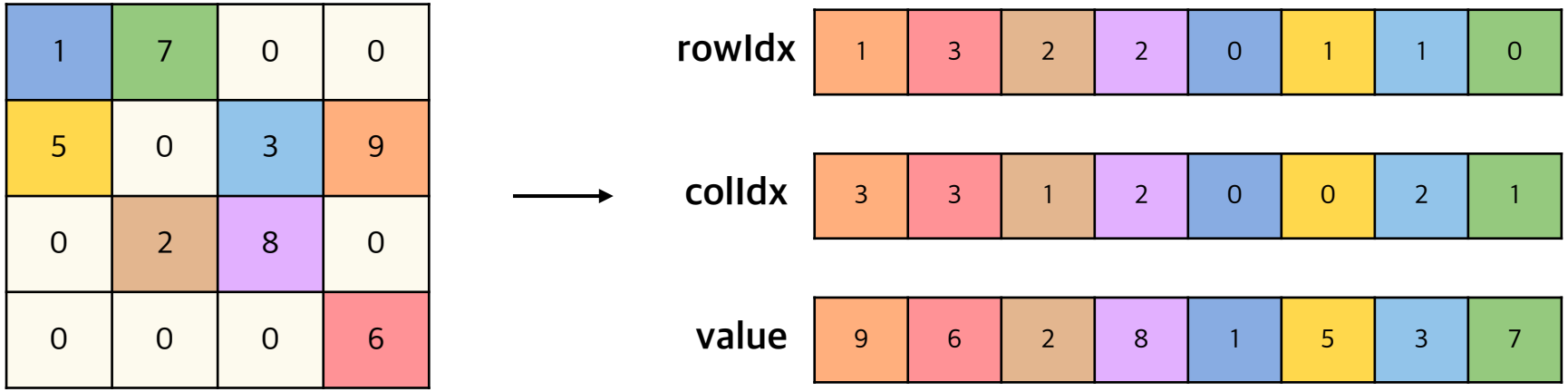
- There are various design considerations that go into the structure of a sparse matrix storage formats.

Space efficiency	The amount of memory capacity that is required to represent the matrix using the storage format.
Flexibility	The extent to which the storage format makes it easy to modify the matrix by adding or removing nonzeros.
Accessibility	The kinds of data that the storage format makes it easy to access.
Memory access efficiency	The extent to which the storage format enables an efficient memory access pattern for a particular computation.
Load balance	The extent to which the storage format balances the load across different threads for a particular computation.

14.2 A simple SpMV kernel with the COO format

Coordinate list (COO) storage format

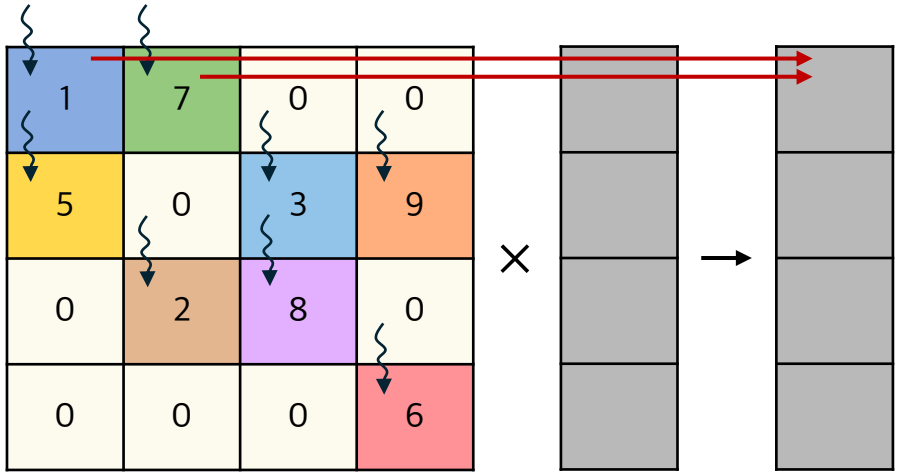
- COO (coordinate list) stores the nonzero values in a one-dimensional array, which is shown as the value array. Each nonzero element is stored with both its column index and its row index. We have both colIdx and rowIdx arrays to accompany the value array.



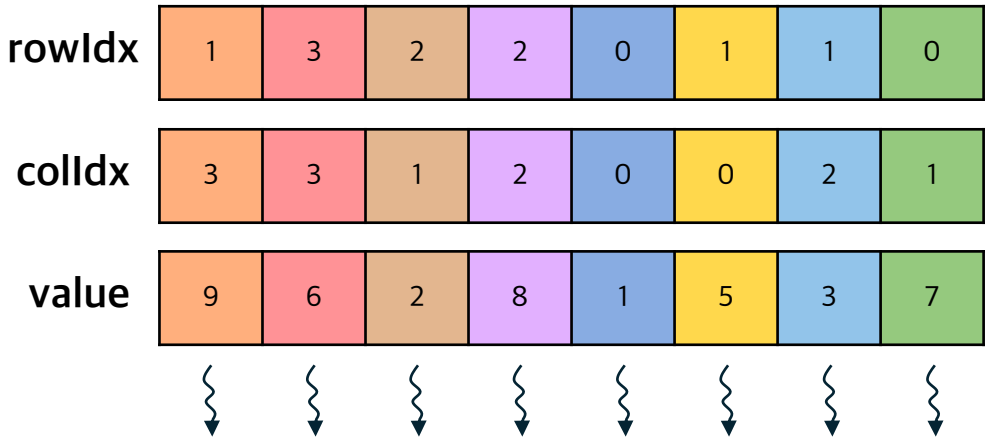
14.2 A simple SpMV kernel with the COO format

SpMV/COO

- One approach to performing SpMV in parallel using a sparse matrix represented in the COO format is to assign a thread to each nonzero element in the matrix.
- An atomic operation is used for the accumulation because multiple threads may update the same output element.



Logical view



Physical view

14.2 A simple SpMV kernel with the COO format

Atomic operation

- An atomic operation on a memory location is an operation that performs a read-modify-write sequence on the memory location in such a way that no other read-modify-write sequence to the location can overlap with it. That is, the read, modify, and write parts of the operation form an undividable unit, hence the name atomic operation.

```
int atomicAdd (int* address, int value);
```

- Other types of atomic operations include subtraction, increment, decrement, minimum, maximum, logical and, and logical or.

14.2 A simple SpMV kernel with the COO format

- The main drawback of SpMV/COO is the need to use **atomic operations**. The reason for using atomic operations is that multiple threads are assigned to nonzeros in the same row and therefore need to update the same output value.

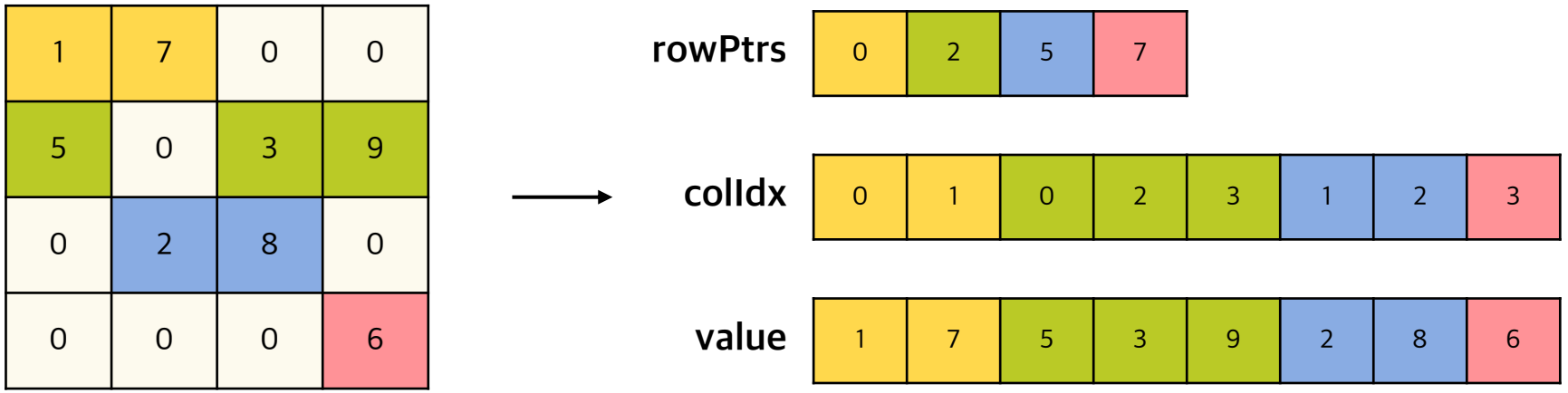
- The atomic operations can be avoided if all the nonzeros in the same row are assigned to the same thread such that the thread will be the only one updating the corresponding output value. However, recall that the COO format does not give this accessibility.

14.3 Grouping row nonzeros with the CSR format

Compressed sparse row (CSR) storage format

- Like the COO format, CSR stores the nonzero values in a one-dimensional array shown as the value array. However, these nonzero values are grouped by row.

- The key distinction between the COO format and the CSR format is that the CSR format replaces the rowIdx array with a rowPtrs array that stores the starting offset of each row's nonzeros in the colIdx and value arrays.

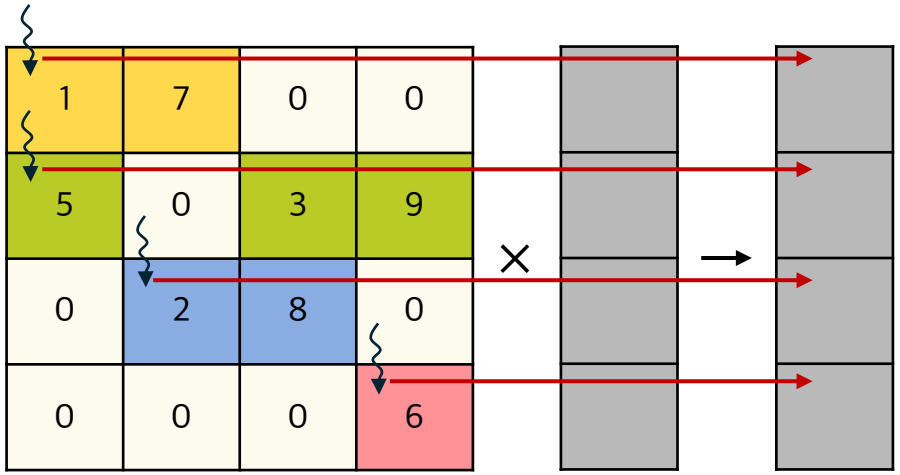


14.3 Grouping row nonzeros with the CSR format

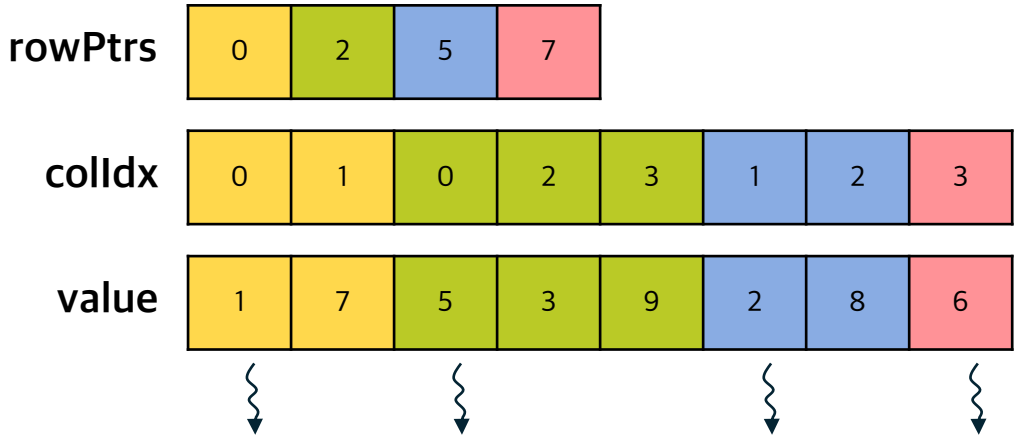
SpMV/CSR

- To perform SpMV in parallel using a sparse matrix represented in the CSR format, one can assign a thread to each row of the matrix.

- The accumulation of the sum to the output vector does not require atomic operations.
  The reason is that each row is traversed by a single thread, so each thread will write to a distinct output value.



Logical view



Physical view

14.3 Grouping row nonzeros with the CSR format

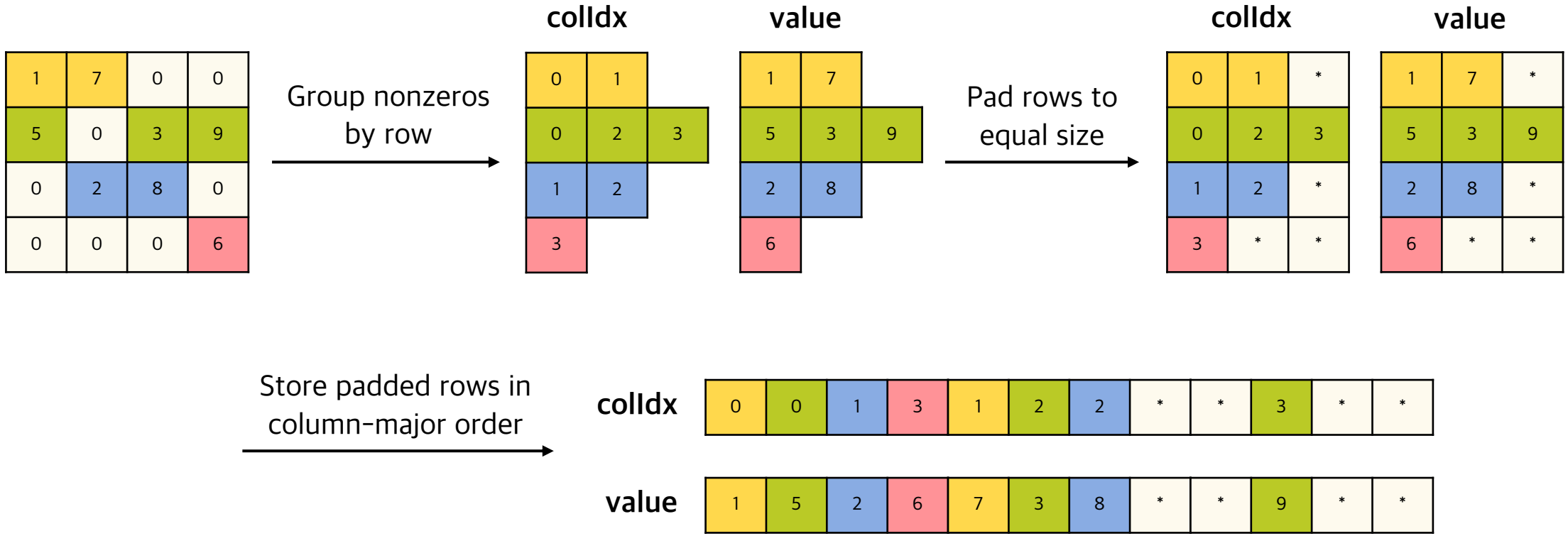
- The advantages of CSR over COO are that it has better space efficiency and that it gives us access to all the nonzeros of a row, allowing us to avoid atomic operations by parallelizing the computation across rows in SpMV/CSR.

- The disadvantages of CSR over COO are that it provides less flexibility with adding nonzero elements to the sparse matrix, it exhibits a memory access pattern that is not amenable to coalescing, and it causes high control divergence.

14.4 Improving memory coalescing with the ELL format

ELL storage format

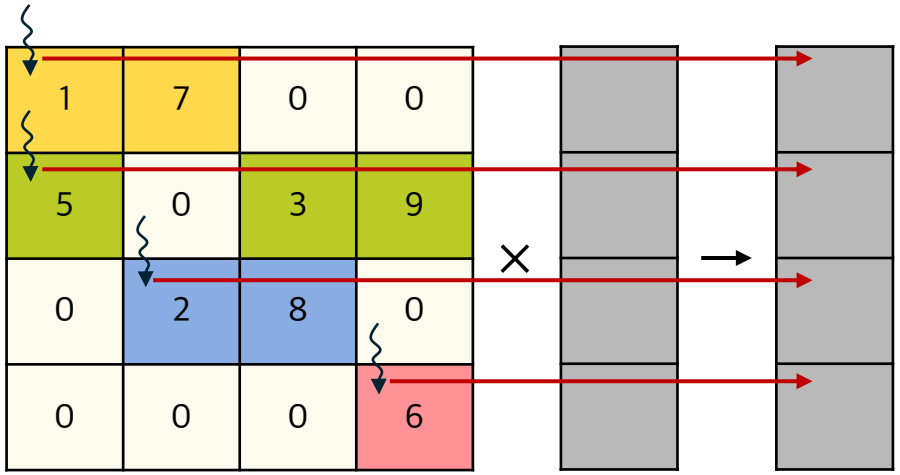
- The problem of noncoalesced memory accesses can be addressed by applying data padding and transposition on the sparse matrix data.



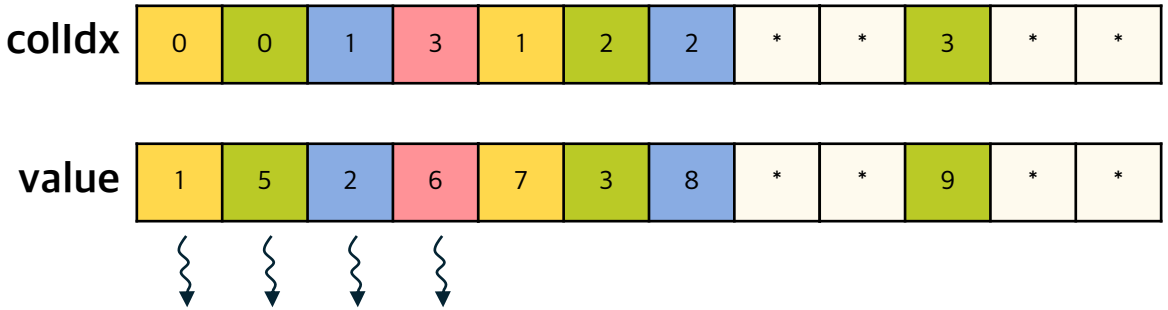
14.4 Improving memory coalescing with the ELL format

SpMV/ELL

- To perform SpMV in parallel using a sparse matrix represented in the ELL format, one can assign a thread to each row of the matrix like CSR format.
- The accesses to the colIdx arrays and the value array are coalesced because consecutive threads have consecutive array indices.



Logical view



Physical view

14.4 Improving memory coalescing with the ELL format

- The ELL format improves on the CSR format by allowing more flexibility to add nonzeros by replacing padding elements and, most important, more opportunities for memory coalescing in SpMV/ELL.

- The ELL format is less space efficient than the CSR format, owing to the space overhead of the padding elements. The overhead of the padding elements highly depends on the distribution of nonzeros in the matrix.

- The control divergence of SpMV/ELL is as bad as that of SpMV/CSR.

14.5 Regulating padding with the hybrid ELL-COO format

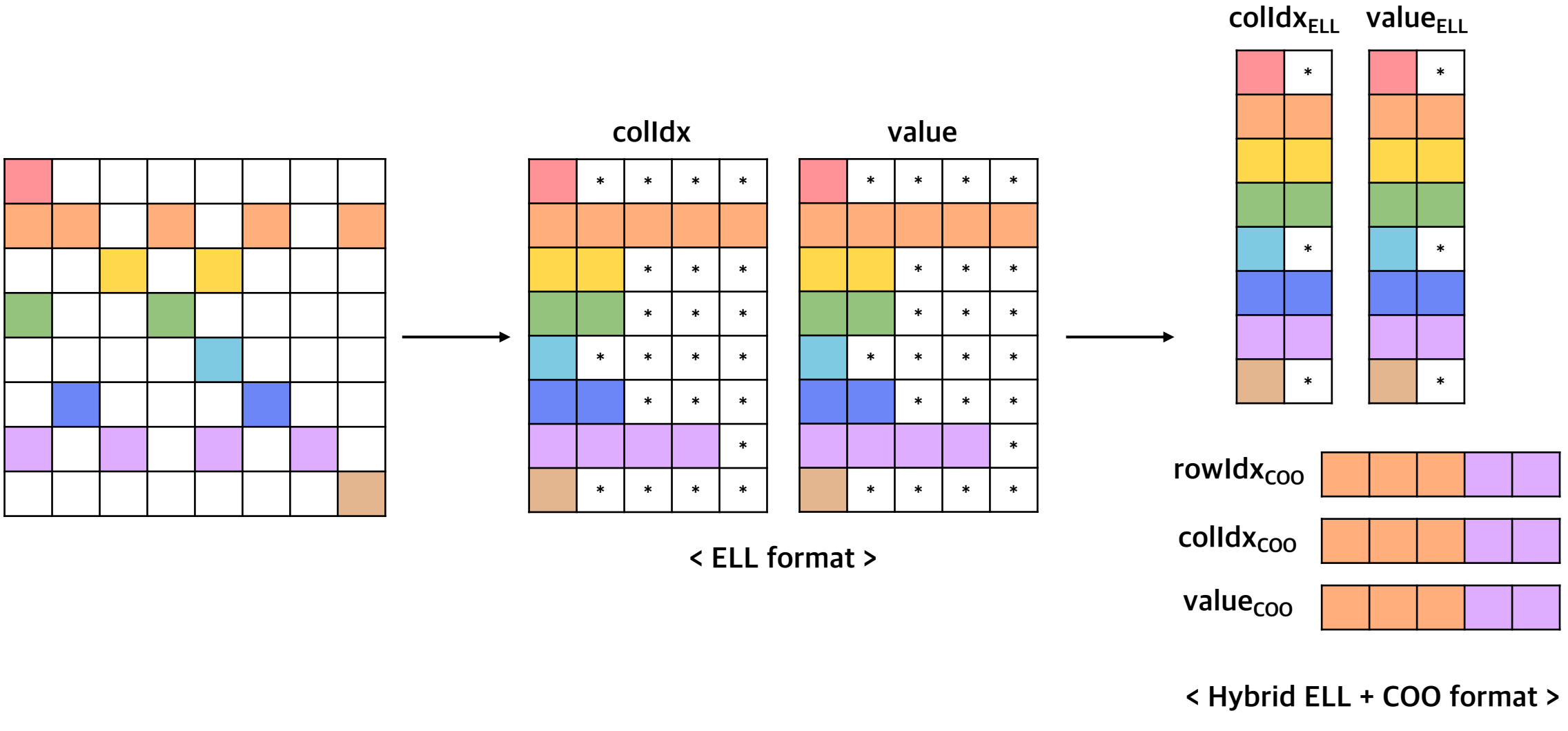
Hybrid ELL-COO format

- The problems of low space efficiency and control divergence in the ELL format are most pronounced when one or a small number of rows have exceedingly large number of nonzero elements.

- The COO format can be used to curb the length of rows in the ELL format. Before we convert a sparse matrix to ELL, we can take away some of the elements from the rows with exceedingly large numbers of nonzero elements and place the elements into a separate COO storage.

- This approach of employing two formats to collaboratively complete a computation is often referred to as a hybrid method.

14.5 Regulating padding with the hybrid ELL-COO format



14.5 Regulating padding with the hybrid ELL-COO format

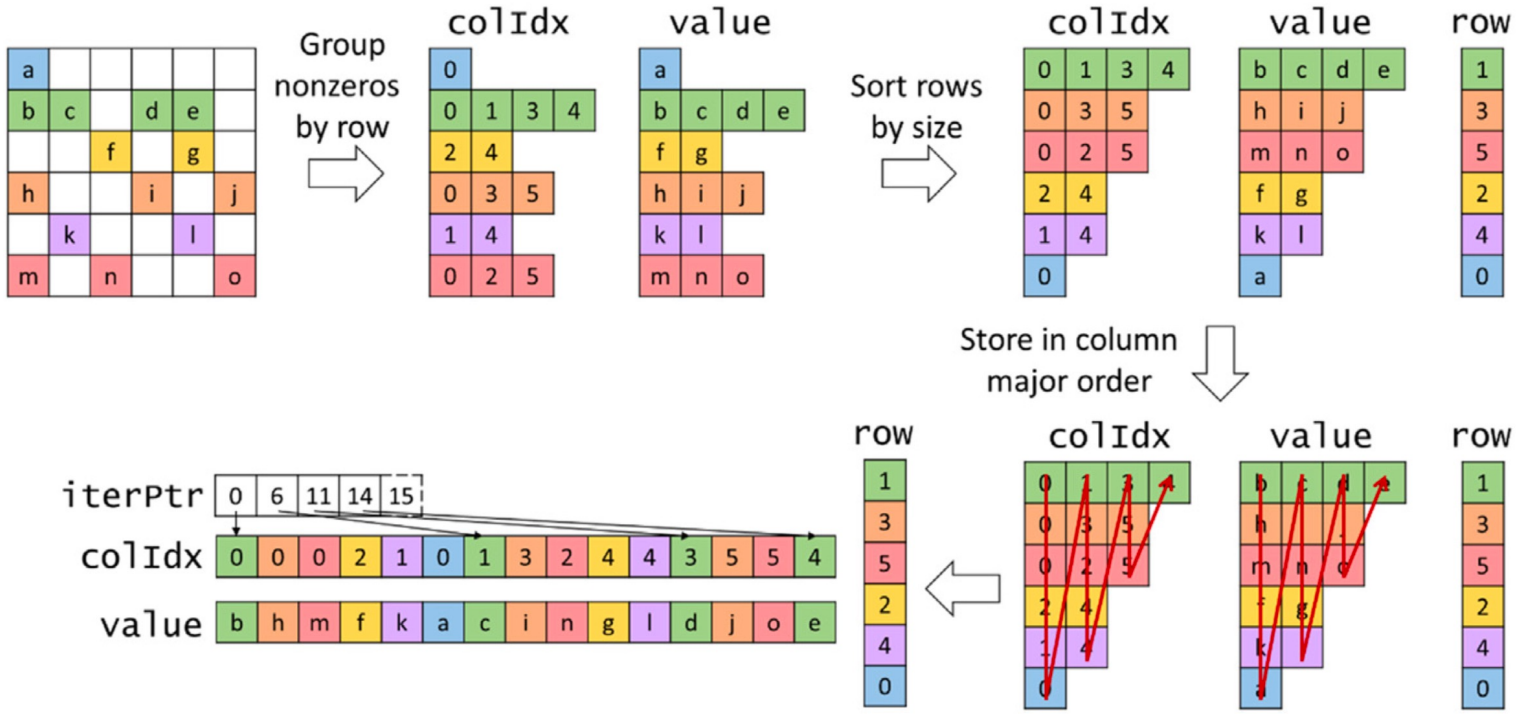
- The hybrid ELL-COO format, in comparison with the ELL format alone, improves space efficiency by reducing padding, provides more flexibility for adding nonzeros to the matrix, retains the coalesced memory access pattern, and reduces control divergence.

- The price that is paid is a small limitation in accessibility, in which it becomes more difficult to access all the nonzeros of a given row if that row overflows to the COO part of the format.

14.6 Reducing control divergence with the JDS format

Jagged diagonal storage (JDS) format

- JDS can achieve coalesced memory access patterns in SpMV and also reduce control divergence without the need to perform any padding. By sorting the rows according to their length, say, from the longest to the shortest.



14.6 Reducing control divergence with the JDS format

- The JDS format is more space efficient than the ELL format because it avoids padding.
  JDS format enables accesses to the sparse matrix to happen in a coalesced manner and it is effective at reducing control divergence.

- The JDS format is less flexible than the CSR format because adding nonzeros changes the sizes of the rows, which may require rows to be resorted.

14.7 Summary

- Methodology

1. Comparing the performance of SpMV algorithms on CPU and GPU.

2. The operation was repeated for 10 times, and the average values were compared.

3. (a) GPU block size: 16x16
   (b) Sparse matrix size:
   1000x1000, 2000x 2000, 3000x 3000
   (c) The ratio of nonzero element in sparse matrix: 1%

Ubuntu Version	Ubuntu 22.04.2 LTS
CUDA Version	12.1

NVIDIA GeForce RTX 3080	
SM Count	68
Max resident threads per SM	1536
Max number of resident blocks per SM	16
Threads in warp	32
Max threads per block	1024
Max thread dimensions	(1024, 1024, 64)
Max grid dimensions	$(2^{31}-1, 2^{16}-1, 2^{16}-1)$
Shared Mem per SM	48 KB
Registers per SM	64 KB
Total constant Mem	64 KB
Total global Mem	10 GB

14.7 Summary

- Results

	CPU Execution Time	GPU Execution Time	GPU (COO format) Execution Time	GPU (CSR format) Execution Time
1000 x 1000	1.193872 (ms)	1.059857 (ms)	0.378393 (ms)	49.537300 (us)
2000 x 2000	4.800317 (ms)	8.204378 (ms)	1.555422 (ms)	0.301598 (ms)
3000 x 3000	10.864206 (ms)	26.574403 (ms)	3.381554 (ms)	1.081664 (ms)