Chapter-10
Parallel patterns: sparse matrix computation
An introduction to data compression and regularization

$$\begin{array}{l} \text{Row 0} \\ \text{Row 1} \\ \text{Row 2} \\ \text{Row 3} \end{array} \quad \begin{vmatrix} 3 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 2 & 4 & 1 \\ 1 & 0 & 0 & 1 \end{vmatrix}$$

FIGURE 10.1: A simple sparse matrix example.

			Row 0		Row 2			Row 3		
Nonzero values	data[7]	{	3,	1,	2,	4,	1,	1,	1	}
Column indices	col_index[7]	{	0,	2,	1,	2,	3,	0,	3	}

Row Pointers	row_ptr[5]	{	0,	2,	2,	5,	7	}

FIGURE 10.2: Example of Compressed Sparse Row (CSR) format.

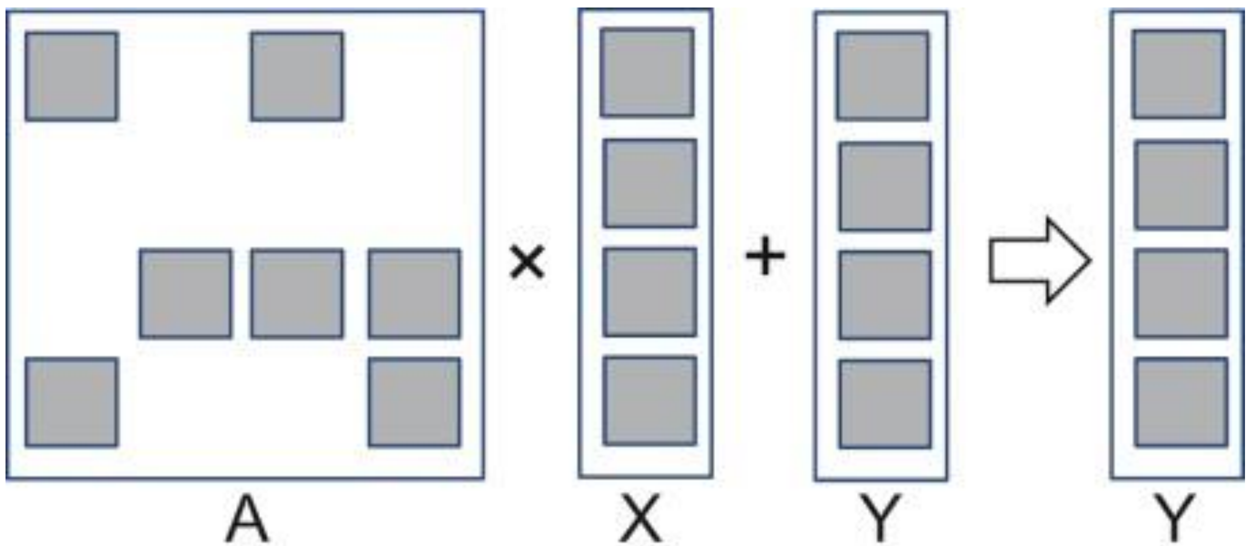

FIGURE 10.3: An example of matrix–vector multiplication and accumulation.

```
for (int row = 0; row < num_rows; row++) {
  float dot = 0;
  int row_start = row_ptr[row];
  int row_end =   row_ptr[row+1];
  for (int elem = row_start; elem < row_end; elem++) {
      dot += data[elem] * x[col_index[elem]];
  }
  y[row] += dot;
}
```

FIGURE 10.4: A sequential loop that implements SpMV based on the CSR format.

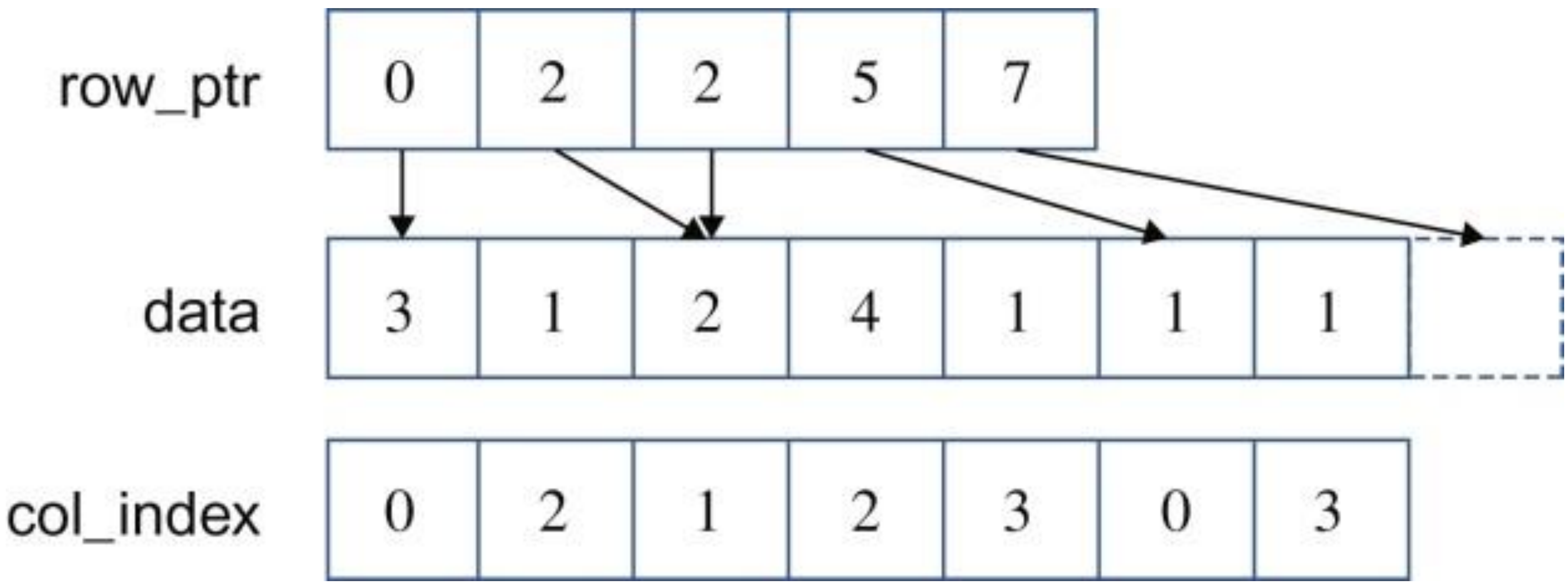

FIGURE 10.5: Illustration of the sequential SpMV loop when operating on the sparse matrix example in Fig. 10.1.

$$\begin{array}{l}\text{Thread 0}\\ \text{Thread 1}\\ \text{Thread 2}\\ \text{Thread 3}\end{array}\left|\begin{array}{cccc}3 & 0 & 1 & 0\\ 0 & 0 & 0 & 0\\ 0 & 2 & 4 & 1\\ 1 & 0 & 0 & 1\end{array}\right|$$

FIGURE 10.6: Example of mapping threads to rows in parallel SpMV/CSR.

```
__global__ void SpMV_CSR(int num_rows, float *data, int *col_index,
  int *row_ptr, float *x, float *y) {

  int row = blockIdx.x * blockDim.x + threadIdx.x;

  if (row < num_rows) {
    float dot = 0;
    int row_start = row_ptr[row];
    int row_end =   row_ptr[row+1];
    for (int elem = row_start; elem < row_end; elem++) {
      dot += data[elem] * x[col_index[elem]];
    }
    y[row] += dot;
  }

}
```

FIGURE 10.7: A parallel SpMV/CSR kernel.

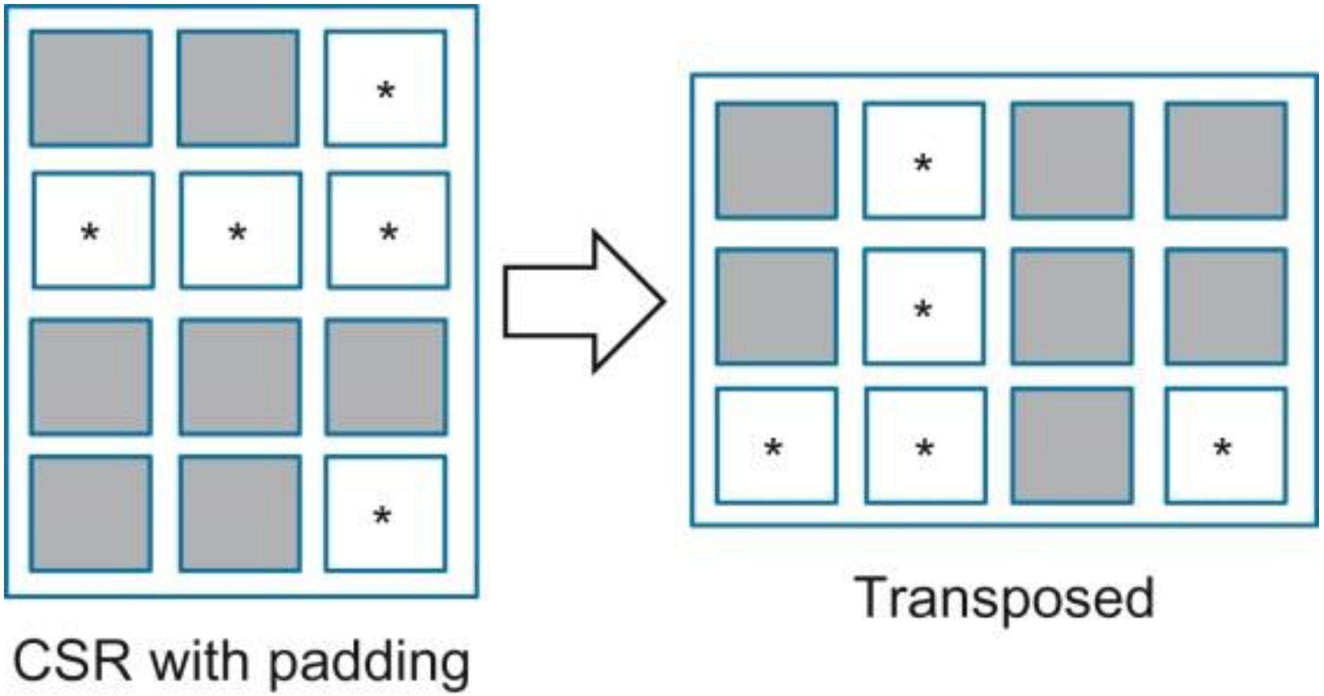

FIGURE 10.8: ELL storage format.

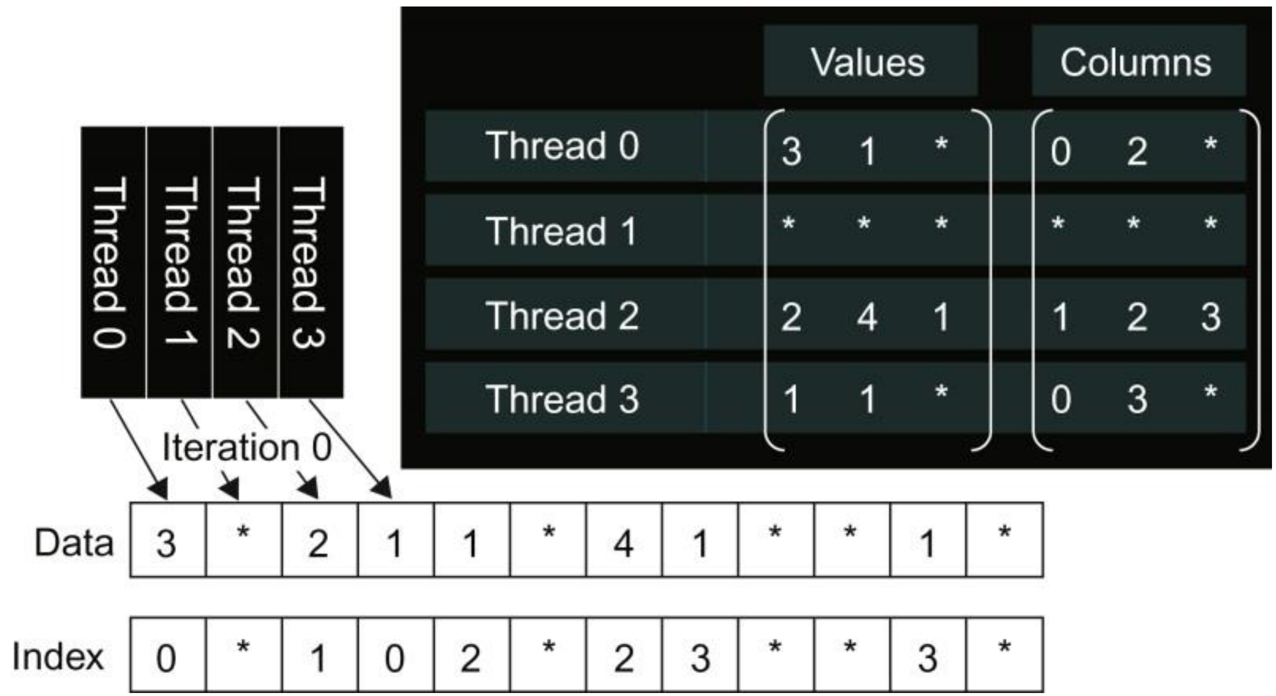

FIGURE 10.9: More details of our small example in ELL.

```
__global__ void SpMV_ELL(int num_rows, float *data, int *col_index,
  int num_elem, float *x, float *y) {

  int row = blockIdx.x * blockDim.x + threadIdx.x;
  if (row < num_rows) {
    float dot = 0;
    for (int i = 0; i < num_elem; i++) {
      dot += data[row+i*num_rows] * x[col_index[row+i*num_rows]];
    }
    y[row] += dot;
  }
}
```

FIGURE 10.10: A parallel SpMV/ELL kernel.

			Row 0	Row 2	Row 3	
Nonzero values	data[7]	{	3, 1,	2, 4, 1,	1, 1	}
Column indices	col_index[7]	{	0, 2,	1, 2, 3,	0, 3	}
Row indices	row_index[7]	{	0, 0,	2, 2, 2,	3, 3	}

FIGURE 10.11: Example of Coordinate (COO) format.

Nonzero values	data[7]	{ 1 1, 2, 4, 3, 1 1 }
Column indices	col_index[7]	{ 0 2, 1, 2, 0, 3, 3 }
Row indices	row_index[7]	{ 3 0, 2, 2, 0, 2, 3 }

FIGURE 10.12: Reordering the Coordinate (COO) format.

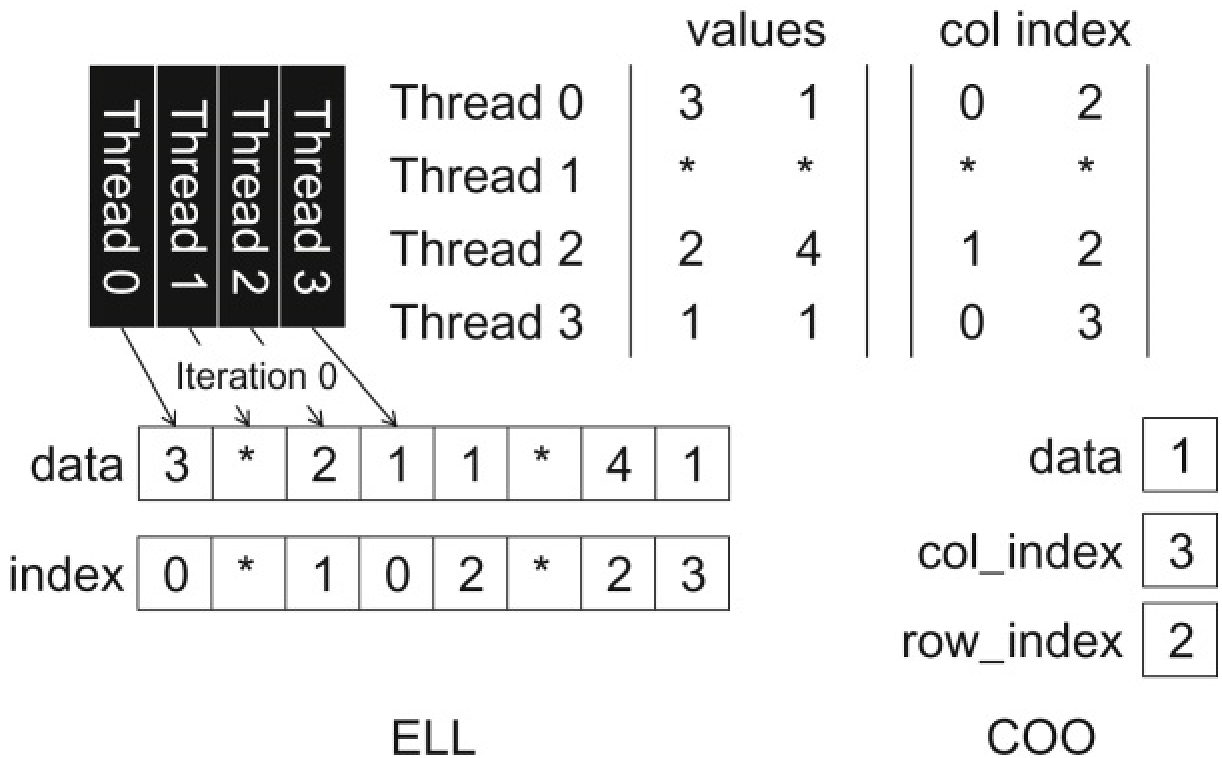

FIGURE 10.13: Our small example in ELL and COO hybrid.

```
for (int i = 0; i < num_elem; row++)
  y[row_index[i]] += data[i] * x[col_index[i]];
```

FIGURE 10.14: A sequential loop that implements SpMV/COO.

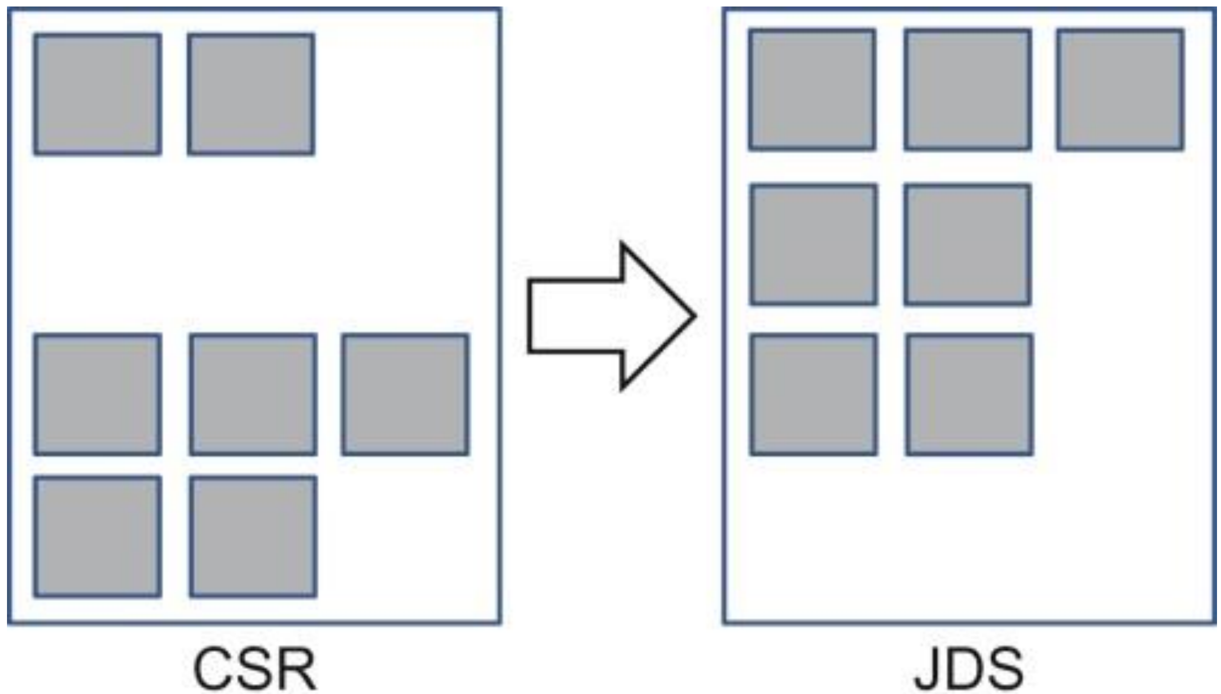

FIGURE 10.15: Sorting rows according to their length.

Nonzero values data[7] { 2, 4, 1, 3, 1, 1, 1 }

Column indices col_index[7] { 1, 2, 3, 0, 2, 0, 3 }

JDS row indices Jds_row_index[4] { 2, 0, 3, 1 }

Section pointers Jds_section_ptr[4] { 0, 3, 7, 7 }

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

FIGURE 10.16: JDS format and sectioned ELL.