GPU Teaching Kit

Accelerated Computing

西南石油大學计算机科学学院
SCHOOL OF COMPUTER SCIENCE, SOUTHWEST PETROLEUM UNIVERSITY

Module 10 – Parallel patterns: Sparse Matrix Computation

Objective

- **Understanding the wealth of work in sparse matrix storage formats and their corresponding parallel algorithms.**
  - Addressing compaction and regularization challenges.
  - Stored format: **CSR**, ELL, COO, Hybrid approach,JDS
  - The parallel SpMV computation performance

GPU Teaching Kit

Accelerated Computing

Module 10 – Parallel patterns: Sparse Matrix Computation

10.1 Background

Objective

- **Understanding what a sparse matrix is along with its storage format and implementing SpMV based on the CSR format.**
  - **Compressed Sparse Row (CSR)** storage format
  - **Implementing SpMV** based on the CSR format.

What we need to know before learning

- **A sparse matrix** is a matrix where **the majority of the elements are zeros.**

- **Sparse matrices** appear in many scientific, engineering, and financial modeling problems.

- Each row of the matrix represents **an equation of a linear system**.

- In various scientific and engineering problems, a large number of equations involve only a small number of variables.

Compressed Sparse Row (CSR) storage format

Sparse Matrix: A

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

		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 }

Row 0 Row 1 Row 2 Row 4

Example of Compressed Sparse Row (CSR) format.

Why We Need Sparse Matrix–Vector Multiplication (SpMV)

– Solving a linear system of N equations of N variables in the form:

$$\mathbf{A*X+Y=0},$$

– where A is an N × N matrix, X is a vector of N variables, and Y is a vector of N constant values. **The objective is to solve for the X variable** that will satisfy all the equations.

– An intuitive(直观) approach is to inverse the matrix such that:

Gaussian elimination

高斯消元法

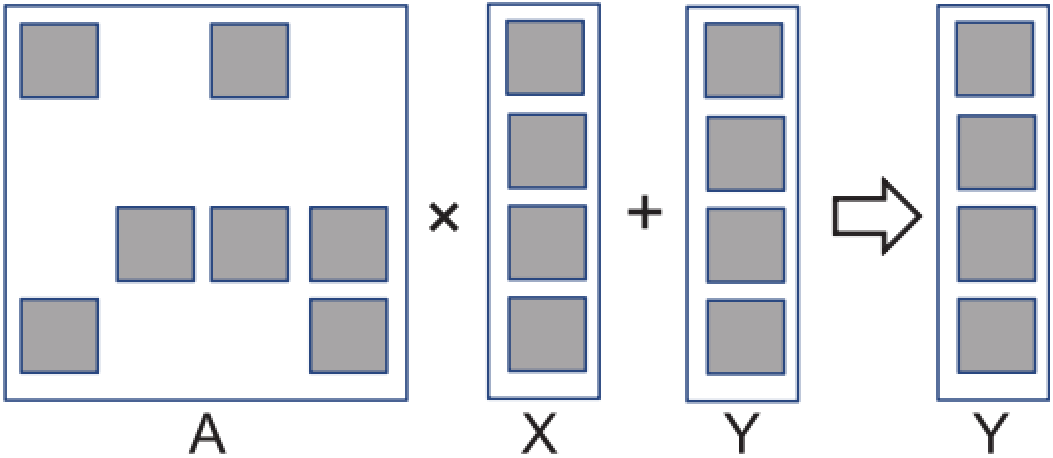
$$\mathbf{X=A^{-1}*(-Y)}$$

– Instead, linear systems of equations represented in sparse matrices can be better solved with an iterative approach:

$$\mathbf{A*X+Y \rightarrow 0}$$

Sparse Matrix–Vector Multiplication (SpMV)

A*X+Y ⟶ Y



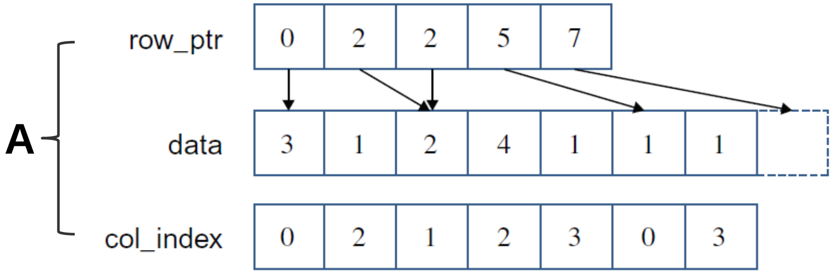
An example of matrix–vector multiplication and accumulation.

A Sequential SpMV/CSR

Sparse Matrix: A

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

```
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;
}
```



X	$\hat{X}$
1	6
2	0
3	20
4	5

NVIDIA

GPU Teaching Kit

Accelerated Computing

Module 10 – Parallel patterns: Sparse Matrix Computation

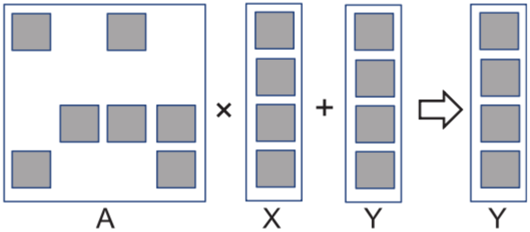
10.2 Parallel SPMV using CSR

Objective

- **Understanding Parallel SPMV using CSR.**
  - Mapping threads to rows.
  - Performance analysis

Mapping Threads to Rows

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

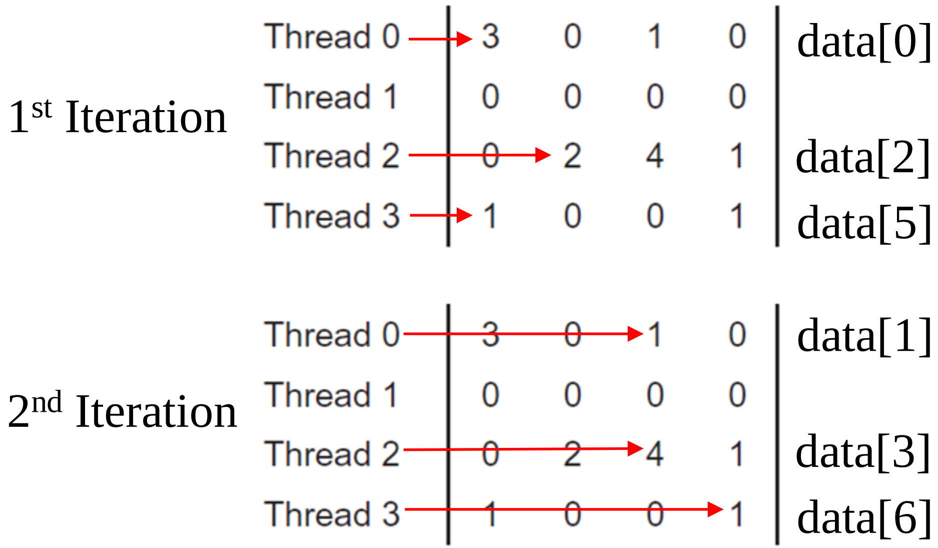


A parallel SpMV/CSR kernel

```
__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;
     }
}
```

Shortcomings of SpMV/CSR Kernel

- **The parallel SpMV/CSR kernel has two major shortcomings**
  - **The kernel does not make <u>coalesced memory accesses</u>.**



Shortcomings of SpMV/CSR Kernel(cont.

- **The parallel SpMV/CSR kernel has two major shortcomings**
  - **its potential to incur significant <u>control flow divergence</u> in all warps.**

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

No. of nonzero elements

Adjacent rows can have varying numbers of nonzero elements

GPU Teaching Kit

Accelerated Computing

Module 10 – Parallel patterns: sparse matrix computation

10.3 Padding and Transposition

Objective

- **Understanding how to use ELL to address the problems.**
  - How to address the problems of **noncoalesced memory accesses** and **control divergence**.
  - **ELL storage format**.

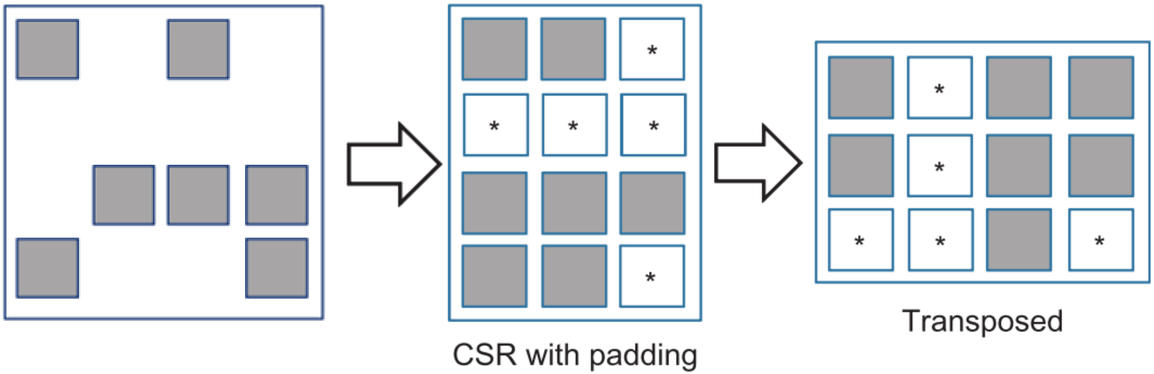
How to solve the two problems

– By applying **<u>data padding</u>** and **<u>transpose</u>** on sparse matrix data.

– These ideas were used in the **ELL** storage format, whose name came from the sparse matrix package in **<u>ELLPACK</u>**, a package for solving elliptic boundary value problems

ELL : PADDING AND TRANSPOSITION

– Determine the row with **<u>the maximum number of non-zero elements</u>**.
– **Add pseudo (zero) elements to all other rows** after non-zero elements, so that they have **the same length as the maximum row**.
– Row priority **transpose** rectangular matrix.



Example of ELL format.

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

Step1:CSR with padding

Values

3	1	*
*	*	*
2	4	1
1	1	*

Columns

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

Step2:transpose

Values

3	*	2	1
1	*	4	1
*	*	1	*

Columns

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

A

data	3	*	2	1	1	*	4	1	*	*	1	*
col_index	0	*	1	0	2	*	2	3	*	*	3	*

ELL

Note: We no longer need row_ ptr array, because **the start of the i-th row (which is now the i-th column in transposed matrix) has been simplified to data [i] through data[num_rows-1].**

Example of ELL format.

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

Step1:CSR with padding

Values

3	1	*
*	*	*
2	4	1
1	1	*

Columns

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

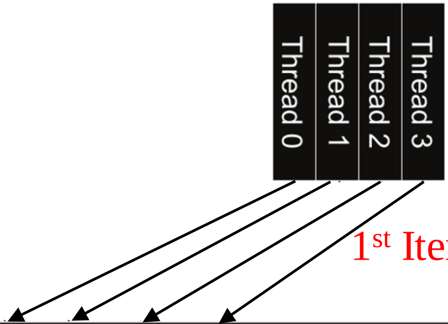
Step2:transpose

Values

3	*	2	1
1	*	4	1
*	*	1	*

Columns

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



1st Iteration

A

data	3	*	2	1	1	*	4	1	*	*	1	*
col_index	0	*	1	0	2	*	2	3	*	*	3	*

ELL

Example of ELL format.

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

Step1:CSR with padding

Values

3	1	*
*	*	*
2	4	1
1	1	*

Columns

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

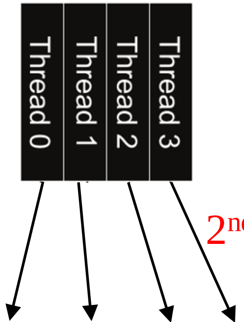
Step2:transpose

Values

3	*	2	1
1	*	4	1
*	*	1	*

Columns

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



2nd Iteration

A

data	3	*	2	1	1	*	4	1	*	*	1	*
col_index	0	*	1	0	2	*	2	3	*	*	3	*

ELL

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

Example of ELL format.

Step1:CSR with padding

Values

3	1	*
*	*	*
2	4	1
1	1	*

Columns

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

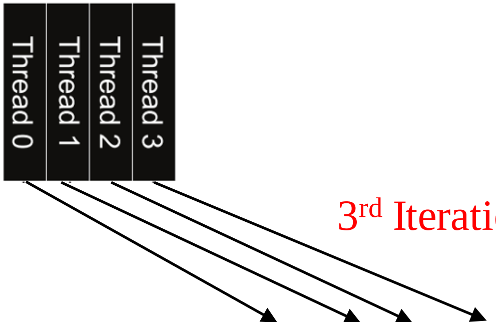
Step2:transpose

Values

3	*	2	1
1	*	4	1
*	*	1	*

Columns

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



3rd Iteration

A

data	3	*	2	1	1	*	4	1	*	*	1	*
col_index	0	*	1	0	2	*	2	3	*	*	3	*

ELL

A parallel SpMV/ELL kernel

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

A

data	3	*	2	1	1	*	4	1	*	*	1	*
col_index	0	*	1	0	2	*	2	3	*	*	3	*

ELL

num_rows=4; //the number of rows in the sparse matrix
num_elem=3; //the maximum number of non-zero elements in all rows of the original sparse matrix.

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

Disadvantages of SpMV/ELL Kernel

- Unfortunately, **SpMV/ELL has a potential drawback**. In cases where one or a few lines have **a significant number of non zero elements**, the ELL format will result in **<u>too many padding elements</u>**.

- This will create a performance bottleneck and storage space issue, which is the problem of uneven thread block load.

How to solve this problem?

GPU Teaching Kit

Accelerated Computing

Module 10 – Parallel patterns: sparse matrix computation

10.4 Using a Hybrid Approach to Regulate Padding

Objective

- **Understanding of how to use a hybrid approach to regulate padding.**
  - The root of the problem with excessive padding in the **ELL** representation is that one or a small number of rows have an exceedingly large number of nonzero elements.
  - **The Coordinate (COO) format** provides such a mechanism.

COO

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

Each non-zero element is stored together with its column index and row index.

		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 }

COO

- **The characteristics of COO:**
  - The elements in COO format can be reordered arbitrarily without losing any information.
  - You can view any element in the storage and know where it comes from the original sparse matrix.

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 }

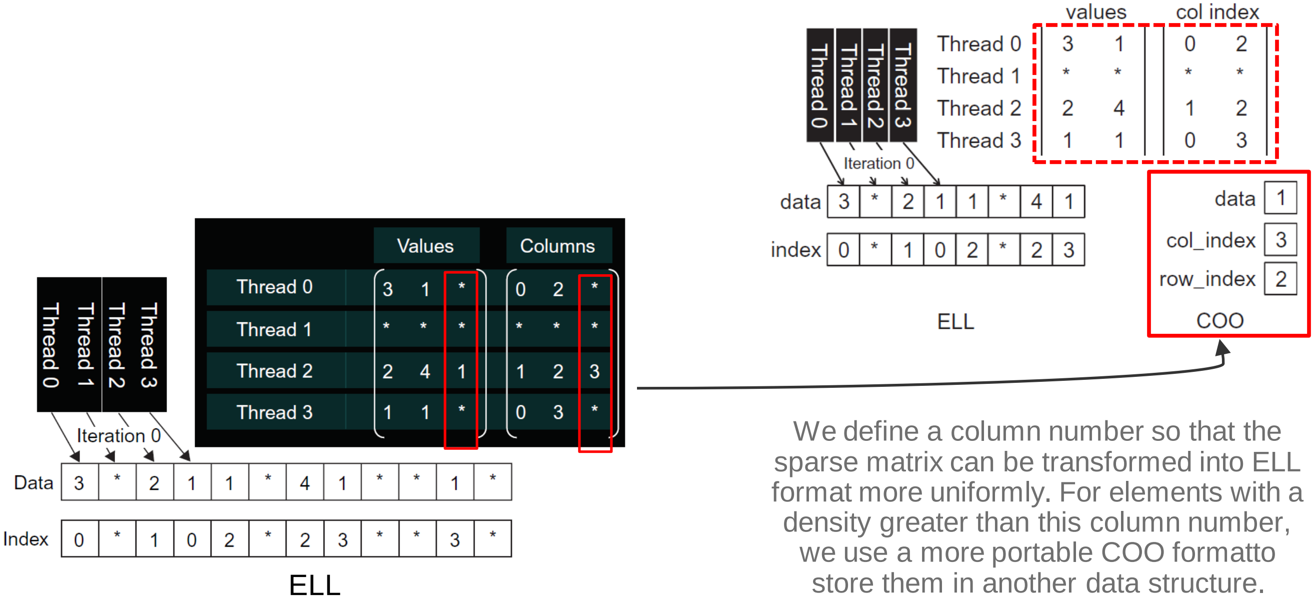
Reordering the COO format.

Limitation: Such reordering would disturb the locality and sequential patterns necessary for the efficient use of memory bandwidth.

Benefit: It can be used to curb the length of rows in the CSR format or the ELL format.

Improve ELL

– **We can remove some elements from rows with a very large number of non-zero elements** and place them in separate **COO** storage. So we can use SpMV/ELL on other elements.



We define a column number so that the sparse matrix can be transformed into ELL format more uniformly. For elements with a density greater than this column number, we use a more portable COO formatto store them in another data structure.

SpMV kernel using ELL-COO hybrid format

```
//SpMV_Hybrid: Sparse Matrix-Vector Multiplication (SpMV) using a hybrid format
__global__ void SpMV_Hybrid(int num_rows,          // Number of rows in the matrix
                            float *data_ell,        // ELL format data array
                            int *col_index_ell,     // ELL format column index array
                            int num_elem_ell,       // Maximum number of non-zero elements
                            float *data_coo,        // COO format data array
                            int *row_index_coo,     // COO format row index array
                            int *col_index_coo,     // COO format column index array
                            int num_elem_coo,       // Number of non-zero elements in COO
                            float *x,               // Dense input vector x
                            float *y) {             // Output vector y
    // Compute the row index this thread is responsible for
    int row = blockIdx.x * blockDim.x + threadIdx.x;

…
```

SpMV kernel using ELL-COO hybrid format

```
//SpMV_Hybrid: Sparse Matrix-Vector Multiplication (SpMV) using a hybrid format
…
  if (row < num_rows) {  // Ensure the thread processes a valid row
    float dot = 0;

    // -------------------------------
    // 1. Process using ELL format
    // -------------------------------
    for (i = 0; i < num_elem_ell; i++) {
      //  Compute the column index of the i-th element in the current row
      int col = col_index_ell[i * num_rows+ row];

      if (col != -1) {// Check if the ELL element is valid (-1 indicates no element)
        // Accumulate the product of the current element and the corresponding
        // vector x element
        dot += data_ell[i * num_rows+ row] * x[col];
      }
    }
```

data_ell

3	*	2	1	1	*	4	1

col_index_ell

0	*	1	0	2	*	2	3

SpMV kernel using ELL-COO hybrid format

```
//SpMV_Hybrid: Sparse Matrix-Vector Multiplication (SpMV) using a hybrid format
…
    // ------------------------------
    // 2. Process using COO format
    // ------------------------------
    for (i = 0; i < num_elem_coo; i++) {
        // Check if the current COO element belongs to this row
        if (row_index_coo[i] == row) {
            dot += data_coo[i] * x[col_index_coo[i]];
        }
    }

    // ------------------------------
    // 3. Write the result back
    // ------------------------------
    y[row] += dot; }
}
```

data_coo	*	*	1	*
row_index_coo	*	*	2	*
col_index_coo	*	*	3	*

GPU Teaching Kit

Accelerated Computing

Module 10 – Parallel patterns: sparse matrix computation

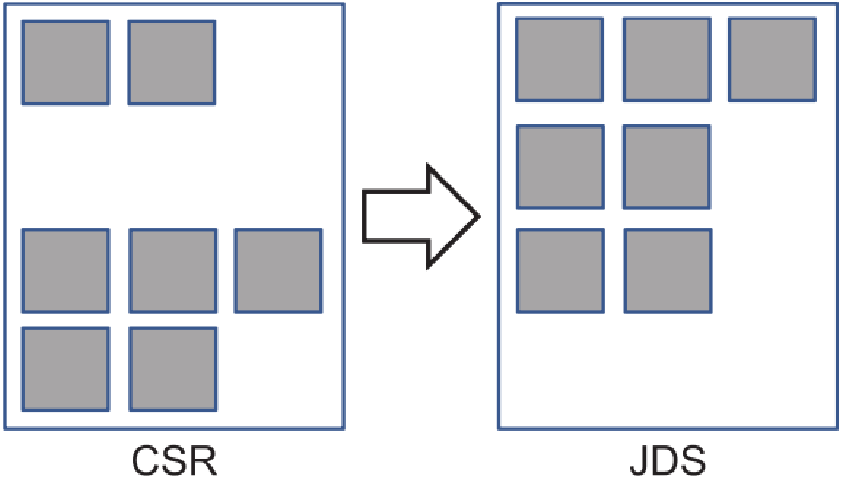
10.5 Sorting and Partitioning for Regularization

Sorting and Partitioning for Regularization

- We can further reduce the filling cost **by sorting and partitioning the rows of sparse matrices**. This idea is to sort rows based on their length.

- The format is often referred to as the Jagged Diagonal Storage (**JDS**) format.

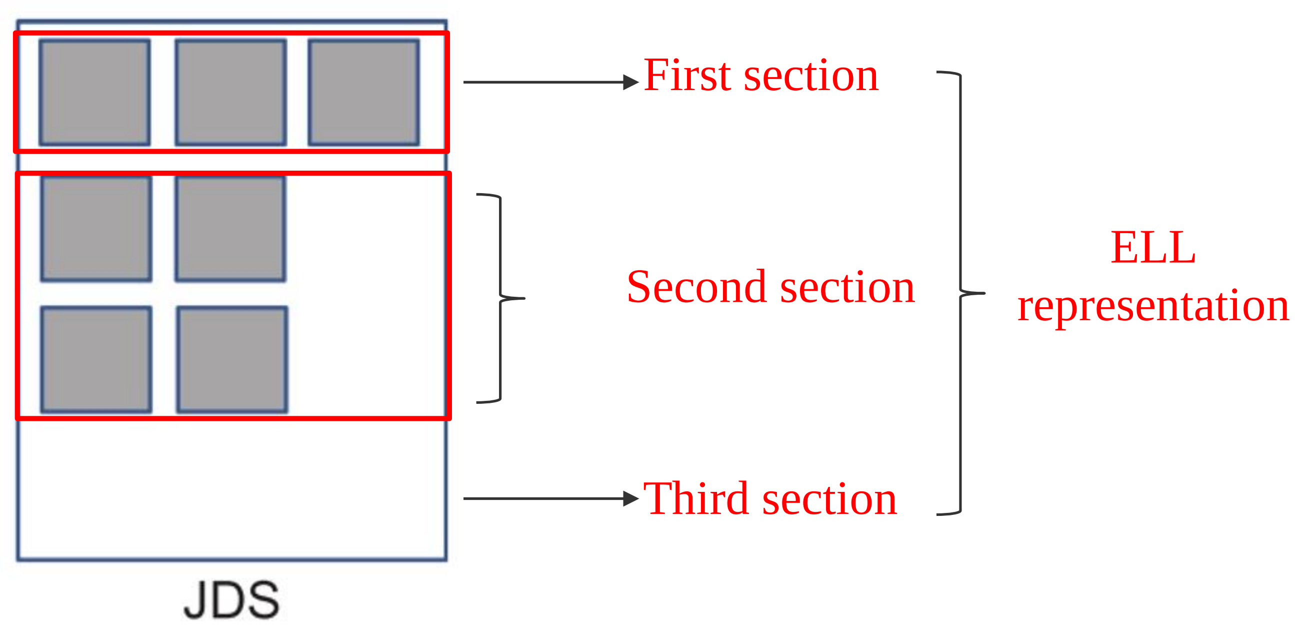
JDS



Sorting rows according to their length.

Benefit of JDS

- Transpose each section independently and launch a separate kernel on each section.
- Do not even need to launch a kernel for the section of rows with no nonzero elements.



JDS format and sectioned ELL

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

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

JDS format and sectioned ELL.

SpMV kernel using JDS format

```
// spmv_jds: Sparse Matrix-Vector Multiplication (SpMV) using JDS format
__global__ void spmv_jds (
    float * data,        // Non-zero values in JDS format
    int * col_index,     // Column indices in JDS format
    int * row_perm,      // Row permutation array(maps rows to original order)
    int * row_nnz,       // The number of non-zero elements per row
    float *  x,          // Dense input vector x
    float *  y,          // Output vector y
    int num_rows         // Total number of rows in the matrix
) {
    // Compute the row index this thread is responsible for
    int row = blockDim.x * blockIdx.x + threadIdx.x;
```

SpMV kernel using JDS format

```
// spmv_jds: Sparse Matrix-Vector Multiplication (SpMV) using JDS format
…
        if (row < num_rows) {  // Ensure the thread processes a valid row
            float dot = 0.0f;
            int row_start = 0;
            // Compute the starting index for this row
            for (int j = 0; j < row; j++) {
                row_start += row_nnz[j];
            }
            // Compute the end index for non-zero elements in this row
            int row_end = row_start + row_nnz[row];
            for (int j = row_start; j < row_end; j++) {
                  dot += data[j] * x[col_index[j]];
            }
           y[row_perm[row]] = dot;
        }
}
```

GPU Teaching Kit

Accelerated Computing

西南石油大學计算机科学学院
SCHOOL OF COMPUTER SCIENCE, SOUTHWEST PETROLEUM UNIVERSITY