Objective

- To learn to regularize irregular data with
  - Limiting variations with clamping
  - Sorting
  - Transposition

- To learn to write a high-performance SpMV
  kernel based on JDS transposed format

Coordinate (COO) format

- Explicitly list the column and row indices for
  every non-zero element

```
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 Allows Reordering of Elements

```
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 }
```

```
Nonzero values data[7] = { 1, 1, 2, 4, 3, 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 }
```
for (int i = 0; i < num_elem; row++)
    y[row_index[i]] += data[i] * x[col_index[i]];

1. a sequential loop that implements SpMV/COO

COO Kernel Design
Accessing Input Matrix and Vector

All threads can access matrix data and the accessed col_index to access vector

Maximal parallelism.

COO kernel Design
Accumulating into Output Vector

Each thread uses the row_index of its element to accumulate into one of the output Y elements

Need atomic operations!

Hybrid Format

- ELL handles typical entries
- COO handles exceptional entries
  - Implemented with segmented reduction

Often implemented in sequential host code in practice
Reduced Padding with Hybrid Format

<table>
<thead>
<tr>
<th>Thread 0</th>
<th>Thread 1</th>
<th>Thread 2</th>
<th>Thread 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>col_index</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 1</td>
<td>0 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* *</td>
<td>* *</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 4</td>
<td>1 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>0 3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Sorting Rows According to Length (Regularization)

CSR to JDS Conversion

JDS (Jagged Diagonal Sparse) Kernel Design for Load Balancing

Sort rows into descending order according to number of non-zero. Keep track of the original row numbers so that the output vector can be generated correctly.

CSR to JDS Conversion
JDS Summary

Nonzero values data[7] { 2, 4, 1, 3, 1, 1, 1 }
Column indices Jds_col_index[7] { 1, 2, 3, 0, 2, 0, 3 }
JDS row indices Jds_row_perm[4] { 2, 0, 3, 1 }
JDS Row Ptrs Jds_row_ptr[5] { 0, 3, 5, 7, 7 }

A Parallel SpMV/JDS Kernel

1. __global__ void SpMV_JDS(int num_rows, float *data, int *col_index, int *jds_row_ptr, int *jds_row_perm, float *x, float *y) {
2.    int row = blockIdx.x * blockDim.x + threadIdx.x;
3.    if (row < num_rows) {
4.      float dot = 0;
5.      int row_start = jds_row_ptr[row];
6.      int row_end = jds_row_ptr[row+1];
7.      for (int elem = row_start; elem < row_end; elem++) {
8.        dot += data[elem] * x[col_index[elem]];
9.      } y[jds_row_perm[row]] = dot;
10.  }

JDS vs. CSR - Control Divergence

- Threads still execute different number of iterations in the JDS kernel for-loop
  - However, neighboring threads tend to execute similar number of iterations because of sorting.
  - Better thread utilization, less control divergence

JDS vs. CSR Memory Divergence

- Adjacent threads still access non-adjacent memory locations

©Wen-mei W. Hwu and David Kirk/NVIDIA, 2010-2018
JDS with Transposition

Access with col_index
Perm with jds_row_index

JDS Format with Transposed Layout

<table>
<thead>
<tr>
<th>Row 0</th>
<th>3</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>Thread 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Thread 1</td>
</tr>
<tr>
<td>Row 2</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>Thread 2</td>
</tr>
<tr>
<td>Row 3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Thread 3</td>
</tr>
</tbody>
</table>

JDS row indices  jds_row_perm[4] { 2, 0, 3, 1 }
JDS column pointers  jds_t_col_ptr[4] { 0, 3, 6, 7 }

data  col_index

2  4  1
3  1
1  1

JDS with Transposition for Memory Coalescing

Transposition for Memory Coalescing

<table>
<thead>
<tr>
<th>2</th>
<th>4</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

JDS row indices  jds_row_perm[4] { 2, 0, 3, 1 }
JDS column pointers  jds_t_col_ptr[4] { 0, 3, 6, 7 }

data  col_index

2  3  1
4  1  1
1  1
1  0  2  2  3  3
### A Parallel SpMV/JDS_T Kernel

```c
1. __global__ void SpMV_JDS_T(int num_rows, float *data,  
   int *col_index, int *jds_t_col_ptr, int *jds_row_perm,  
   float *x, float *y) {  
2.    int row = blockIdx.x * blockDim.x + threadIdx.x;  
3.    if (row < num_rows) {  
4.      float dot = 0;  
5.      unsigned int sec = 0;  
6.      while (jds_t_col_ptr[sec+1]-jds_t_col_ptr[sec] > row){  
7.        sec++;  
8.        dot += data[jds_t_col_ptr[sec]+row] *  
9.          x[col_index[jds_t_col_ptr[sec]+row]];  
10.       sec++;  
11.      }  
12.      y[jds_row_perm[row]] = dot;  
13.    }  
```

### MP7 Variable Names

- **JDS_T Length of Cols** `matRows[4]`:
  - `Sec 0`: 3, 2, 2, 0
  - `Sec 1`: 2, 4, 1, 1
  - `Sec 2`: 1

- **Nonzero values** `matData[7]`:
  - `Sec 0`: 2, 3, 1, 4, 1, 1

- **Column indices** `matCol[7]`:
  - `Sec 0`: 1, 0, 3, 2, 2, 3

- **JDS_T Column Pointers** `matColStart[4]`:
  - `Sec 0`: 0, 3, 6, 7

- **JDS Row Indices** `matRowPerm[4]`:
  - `Sec 0`: 2, 0, 3, 1

---

**JDS with Transposition Memory Coalescing**

<table>
<thead>
<tr>
<th>Thread 0</th>
<th>Thread 1</th>
<th>Thread 2</th>
<th>Thread 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **data**
  - 2, 3, 1, 4, 1, 1, 1

- **col_index**
  - 1, 0, 0, 2, 2, 3, 3

Not aligned with DRAM bursts but OK with recent GPUs
Sparse Matrices as Foundation for Advanced Algorithm Techniques

- Graphs are often represented as sparse adjacency matrices
  - Used extensively in social network analytics, natural language processing, etc.
- Binning techniques often use sparse matrices for data compaction
  - Used extensively in ray tracing, particle-based fluid dynamics methods, and games
- These will be covered in ECE508/CS508

ANY QUESTIONS?