Sparse Matrix-Vector Multiplication

CUDA Implementation

Sukjoon Oh

Contents

1 Sparse Matrix-Vector Multiplication (SpMV)

Sparse Matrix : Matrix in which most of the elements are zero.

SpMV **: Sp**arse **M**atrix-**V**ector (Multiplication)

$$y = Ax$$

Sparse Matrix

- Usual matrix formats (dense formats) are not efficient for implementing kernel
- Memory is wasted for storing zero values and the computing power lost in many multiplications by zero.

Sparse Matrix Representation

References

Bell, N., & Garland, M. (2008). *Efficient sparse matrix-vector multiplication on CUDA* (Vol. 2, No. 5). Nvidia Technical Report NVR-2008-004, Nvidia Corporation.

Bell, N., & Garland, M. (2009, November). Implementing sparse matrix-vector multiplication on throughput-oriented processors. In *Proceedings of the conference on high performance computing networking, storage and analysis* (pp. 1-11).

Benatia, A., Ji, W., Wang, Y., & Shi, F. (2018). BestSF: a sparse meta-format for optimizing SpMV on GPU. *ACM Transactions on Architecture and Code Optimization (TACO)*, *15*(3), 1-27.

Saad, Y. (2003). *Iterative methods for sparse linear systems*. Society for Industrial and Applied Mathematics. P.92-

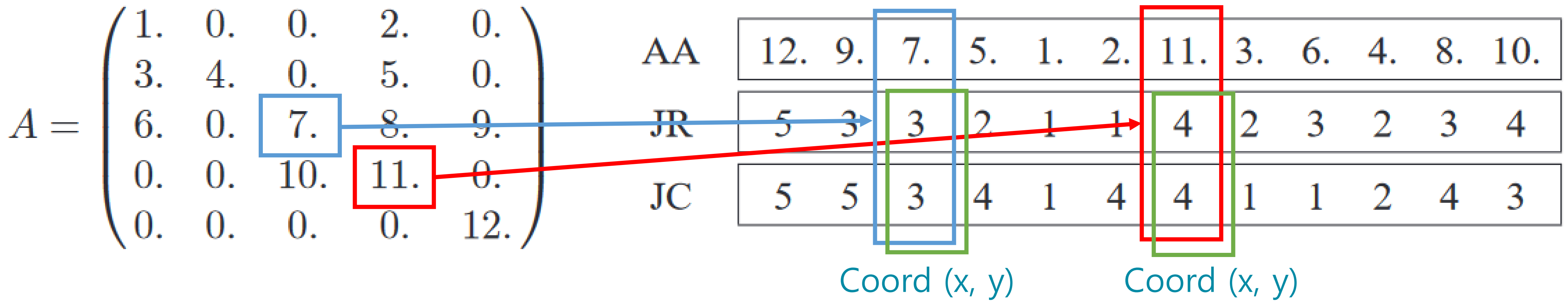
Sparse Matrix Representation

1. Coordinate Format (COO)

- (1) Real array containing all the real (or complex) values of the nonzero elements of A in any order.
- (2) An integer array containing their row indices.
- (3) Second integer array containing their column indices.
- All three arrays are of length N_z (Number of nonzero elements)

Sparse Matrix Representation

1. Coordinate Format (COO)



$$A = \begin{pmatrix} 1. & 0. & 0. & 2. & 0. \\ 3. & 4. & 0. & 5. & 0. \\ 6. & 0. & \boxed{7.} & 8. & 9. \\ 0. & 0. & 10. & \boxed{11.} & 0. \\ 0. & 0. & 0. & 0. & 12. \end{pmatrix}$$

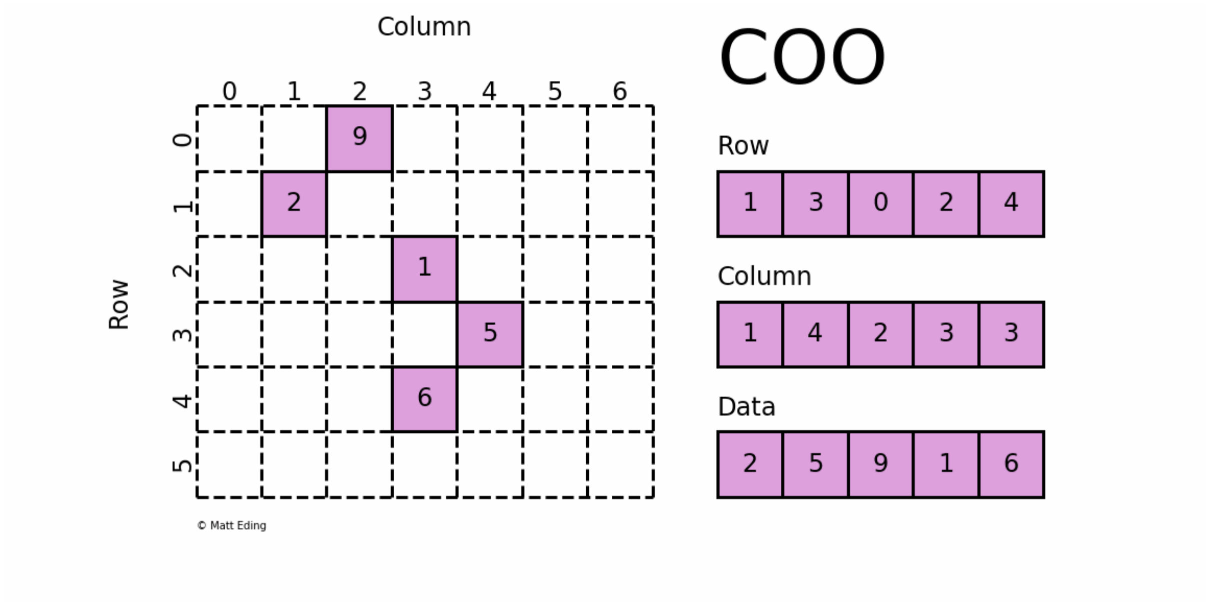
AA	12.	9.	7.	5.	1.	2.	11.	3.	6.	4.	8.	10.
JR	5	3	3	2	1	1	4	2	3	2	3	4
JC	5	5	3	4	1	4	4	1	1	2	4	3

- Elements are usually listed by row or columns.

2

Sparse Matrix Representation

1. Coordinate Format (COO)



- If the elements were listed by row, the array JC might be replaced by an array which points to the beginning of each row instead.

Sparse Matrix Representation

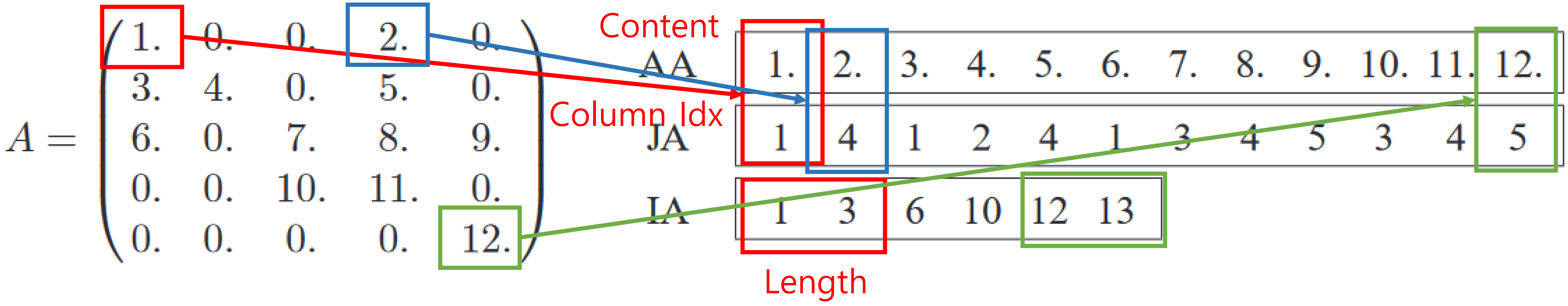
2. Compressed Sparse Row (CSR)

- (1) A real array AA contains the real values a_{ij} stored row by row, from row 1 to n. The length of $\underline{AA}$ is $\underline{N_z}$.
- (2) An integer array JA contains the column indices of the elements a_{ij} as stored in the array AA. The length of JA is N_z.
- (3) An integer array IA contains the pointers to the beginning of each row in the arrays AA and JA.

2

Sparse Matrix Representation

2. Compressed Sparse Row (CSR)



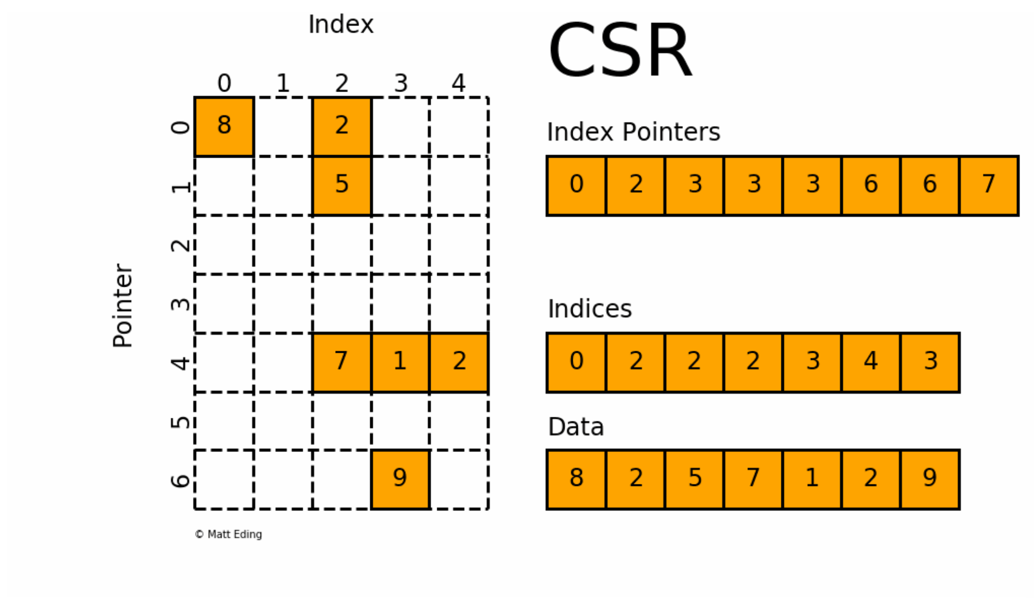
$$A = \begin{pmatrix} 1. & 0. & 0. & 2. & 0. \\ 3. & 4. & 0. & 5. & 0. \\ 6. & 0. & 7. & 8. & 9. \\ 0. & 0. & 10. & 11. & 0. \\ 0. & 0. & 0. & 0. & 12. \end{pmatrix}$$

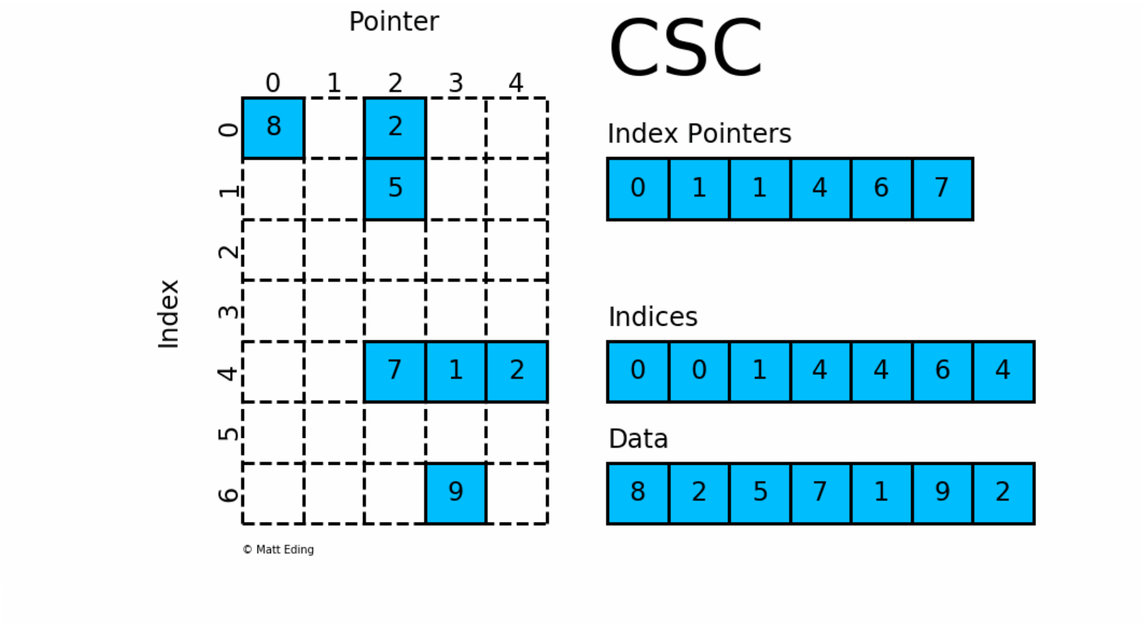
AA	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.	12.
JA	1	4	1	2	4	1	3	4	5	3	4	5
IA	1	3	6	10	12	13						

- Content of $IA(i)$ is the position in arrays AA and JA where the i^{th} row starts.
- Length of $IA(i)$ is $n + 1$
- n is row number

Sparse Matrix Representation

2. CSR Variations





- Compressed Sparse Column (CSC)
- Block Compressed Sparse Row (BCSR)

Sparse Matrix Representation

3. Diagonal Format (DIA)

- Appropriate representation. when nonzero values are restricted to a small number of matrix diagonals.
- Not general-purpose format.
- (1) Rectangular array $DIAG(1{:}n, 1{:}N_d)$, where N_d is the number of diagonals.
- (2) The offsets of each of the diagonals will be stored in an array $IOFF(1{:}N_d)$.

Sparse Matrix Representation

3. Diagonal Format (DIA)

$$A = \begin{pmatrix} 1. & 0. & 2. & 0. & 0. \\ 3. & 4. & 0. & 5. & 0. \\ 0. & 6. & 7. & 0. & 8. \\ 0. & 0. & 9. & 10. & 0. \\ 0. & 0. & 0. & 11. & 12. \end{pmatrix}$$

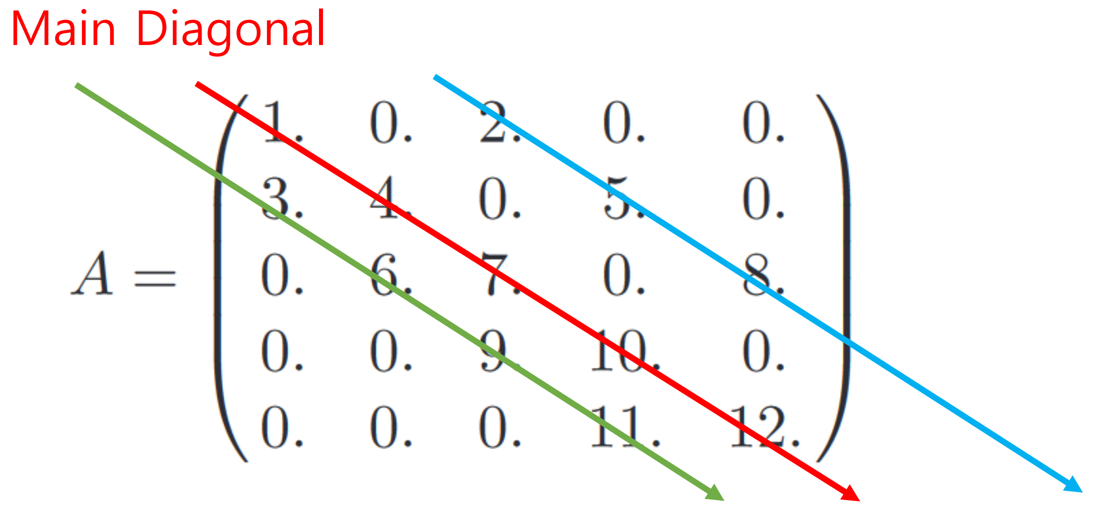
$$\mathtt{DIAG} = \boxed{\begin{array}{ccc} * & 1. & 2. \\ 3. & 4. & 5. \\ 6. & 7. & 8. \\ 9. & 10. & * \\ 11 & 12. & * \end{array}}$$

$$\mathtt{IOFF} = \boxed{\begin{array}{ccc} -1 & 0 & 2 \end{array}}$$

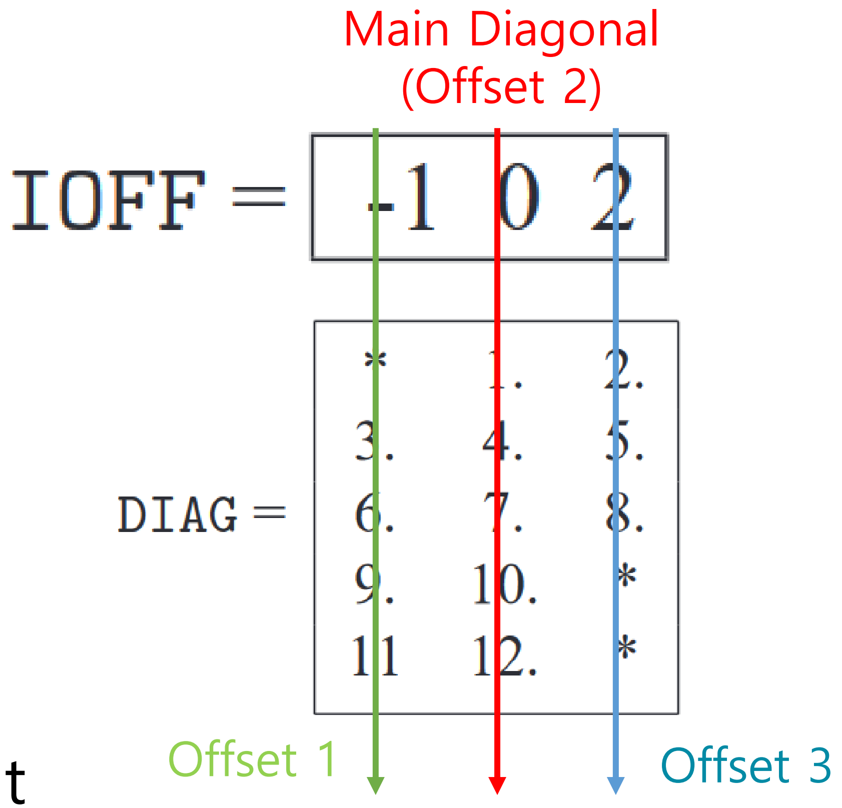
$$DIAG(i,j) \leftarrow a_{i,i+ioff(j)}$$

Sparse Matrix Representation

3. Diagonal Format (DIA)



$$A = \begin{pmatrix} 1. & 0. & 2. & 0. & 0. \\ 3. & 4. & 0. & 5. & 0. \\ 0. & 6. & 7. & 0. & 8. \\ 0. & 0. & 9. & 10. & 0. \\ 0. & 0. & 0. & 11. & 12. \end{pmatrix}$$



$$\text{IOFF} = \boxed{-1 \quad 0 \quad 2}$$

$$\text{DIAG} = \begin{array}{|ccc|} \hline * & 1. & 2. \\ 3. & 4. & 5. \\ 6. & 7. & 8. \\ 9. & 10. & * \\ 11 & 12. & * \\ \hline \end{array}$$

- Zero implies the main diagonal element

Sparse Matrix Representation

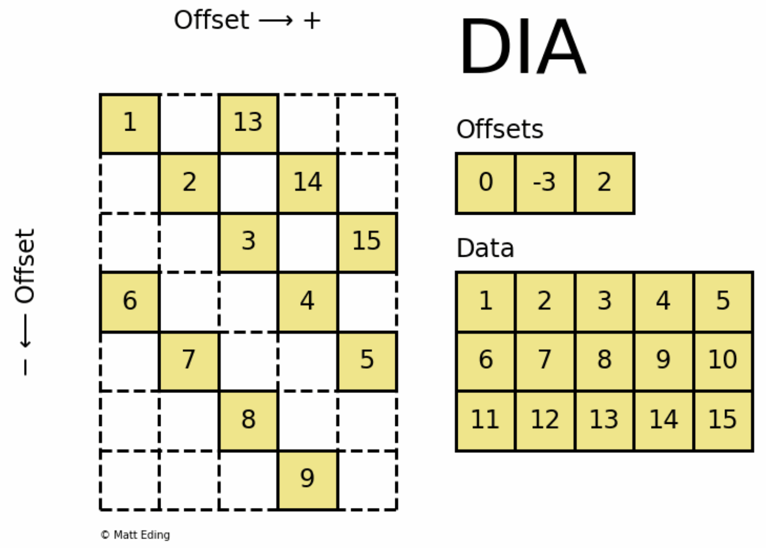
3. Diagonal Format (DIA)

$$
\begin{array}{c}
\ *\ \\
\end{array}
A = \begin{pmatrix} 1. & 0. & 2. & 0. & 0. \\ 3. & 4. & 0. & 5. & 0. \\ 0. & 6. & 7. & 0. & 8. \\ 0. & 0. & 9. & 10. & 0. \\ 0. & 0. & 0. & 11. & 12. \end{pmatrix}
\begin{array}{cc} & \\ & \\ & \\ * & \\ 0 & * \end{array}
\qquad
\mathtt{DIAG} = \boxed{\begin{array}{ccc} * & 1. & 2. \\ 3. & 4. & 5. \\ 6. & 7. & 8. \\ 9. & 10. & * \\ 11 & 12. & * \end{array}}
\qquad
\mathtt{IOFF} = \boxed{-1 \ \ 0 \ \ 2}
$$

- Zero implies the main diagonal element

2 Sparse Matrix Representation

3. Diagonal Format (DIA)



- Implicit indexing reduces the memory footprint and decreases the amount of data transferred during a SpMV operation.
- All memory access to data, x and y is contiguous, which improves the efficiency of memory transactions.

2

Sparse Matrix Representation

3. Diagonal Format (DIA)

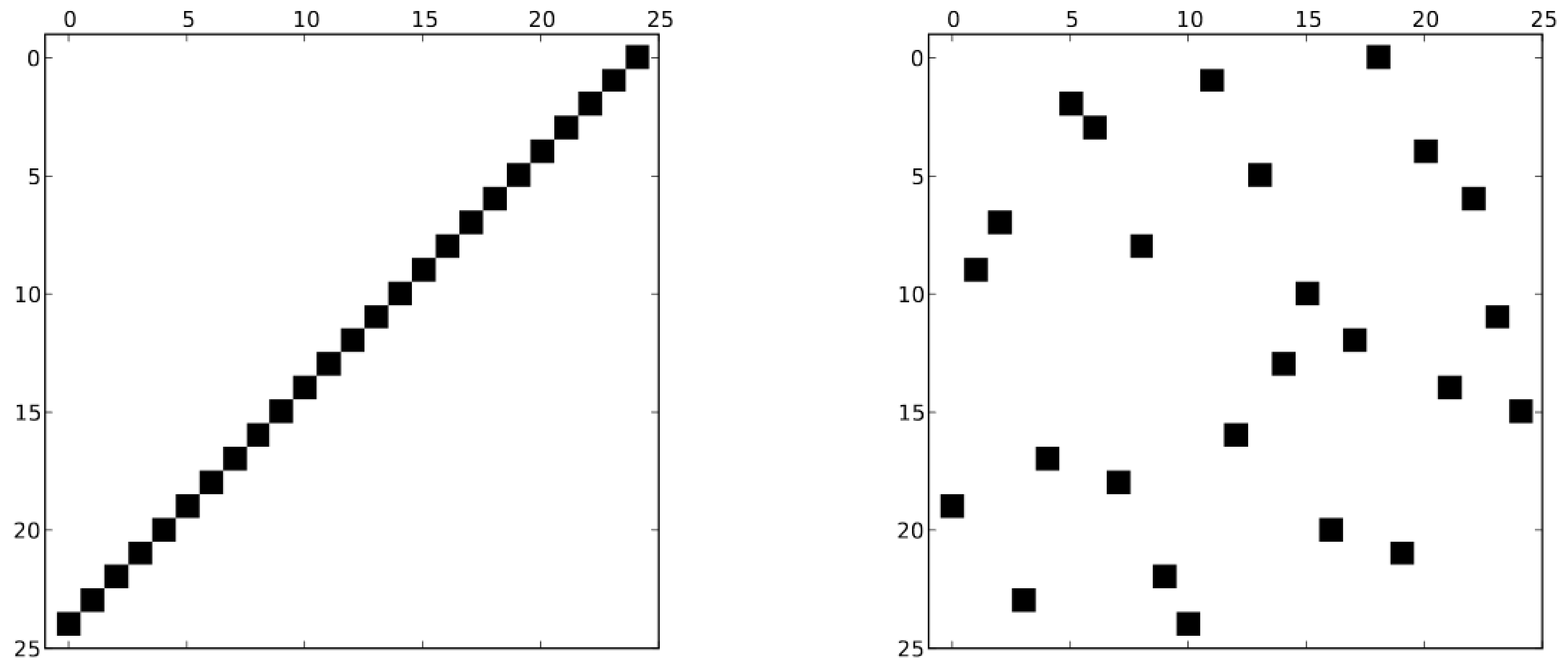


Figure 6: Sparsity patterns that are ill-suited to the sparse diagonal format.

Sparse Matrix Representation

4. Ellpack-Itpack Format (ELL)

- The assumption in this scheme is that there are at most N_d nonzero elements per row, where N_d is small.
- (1) $COEF$, is similar to $DIAG$ and contains the nonzero elements of A. The nonzero elements of each row of the matrix can be stored in a row of the array $COEF(1{:}n, 1{:}N_d)$.
- (2) $JCOEF(1{:}n, 1{:}N_d)$ must store the column positions of each entry in $COEF$.

Sparse Matrix Representation

4. Ellpack-Itpack Format (ELL)

$$A = \begin{array}{c} \begin{array}{ccccc} 1 & & 3 & & \end{array} \\ \begin{pmatrix} \boxed{1. \quad 0. \quad 2. \quad 0. \quad 0.} \\ 3. & 4. & 0. & 5. & 0. \\ 0. & 6. & 7. & 0. & 8. \\ 0. & 0. & 9. & 10. & 0. \\ 0. & 0. & 0. & 11. & 12. \end{pmatrix} \end{array}$$

$$\begin{array}{c}\text{COEF} = \\ \textit{DIAG}\end{array} \begin{array}{|ccc|} \hline \boxed{1. \quad 2. \quad 0.} \\ 3. & 4. & 5. \\ 6. & 7. & 8. \\ 9. & 10. & 0. \\ 11 & 12. & 0. \\ \hline \end{array} \qquad \text{JCOEF} = \begin{array}{|ccc|} \hline \boxed{1 \quad 3} & & 1 \\ 1 & 2 & 4 \\ 2 & 3 & 5 \\ 3 & 4 & 4 \\ 4 & 5 & 5 \\ \hline \end{array}$$

- In this example, those integers are selected to be equal to the row numbers

2 Sparse Matrix Representation

4. Ellpack-Itpack Format (ELL)

$$A = \begin{array}{c} \begin{array}{ccccc} 1 & 2 & & 4 & \end{array} \\ \begin{pmatrix} 1. & 0. & 2. & 0. & 0. \\ 3. & 4. & 0. & 5. & 0. \\ 0. & 6. & 7. & 0. & 8. \\ 0. & 0. & 9. & 10. & 0. \\ 0. & 0. & 0. & 11. & 12. \end{pmatrix} \end{array}$$

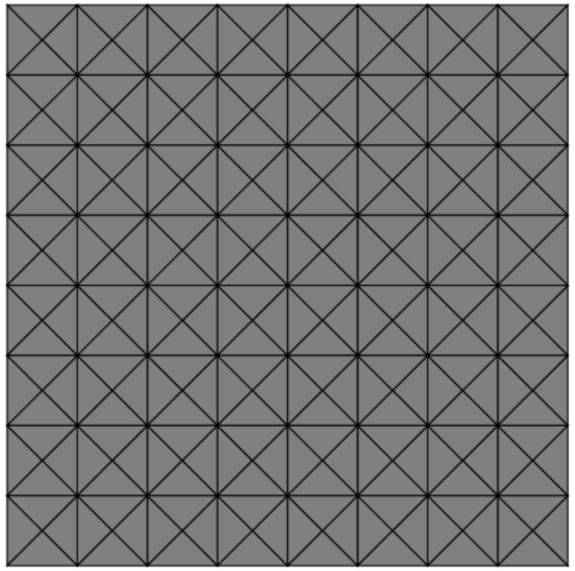
$$\begin{array}{c} \text{COEF} = \\ \textit{DIAG} \end{array} \begin{array}{|ccc|} \hline 1. & 2. & 0. \\ 3. & 4. & 5. \\ 6. & 7. & 8. \\ 9. & 10. & 0. \\ 11 & 12. & 0. \\ \hline \end{array} \qquad \text{JCOEF} = \begin{array}{|ccc|} \hline 1 & 3 & 1 \\ 1 & 2 & 4 \\ 2 & 3 & 5 \\ 3 & 4 & 4 \\ 4 & 5 & 5 \\ \hline \end{array}$$

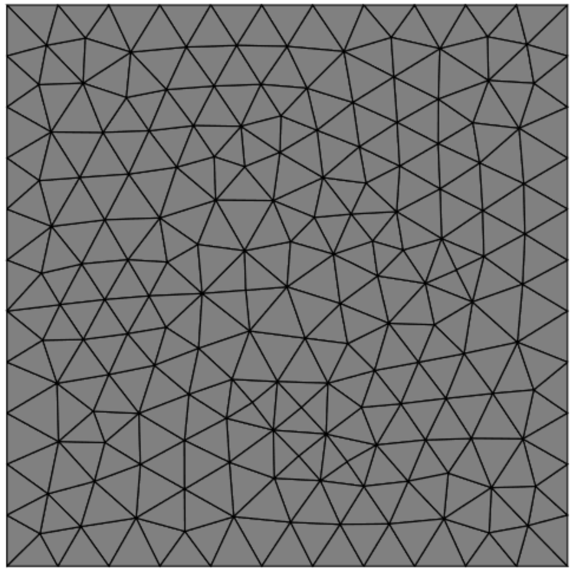
- In this example, those integers are selected to be equal to the row numbers

Sparse Matrix Representation

4. Ellpack-Itpack Format (ELL)

- ELL is more general than DIA since the nonzero columns need not follow any particular pattern.
- The maximum vertex degree is not significantly greater than the average degree.
- In practice, unstructured meshes do not always meet this requirement.

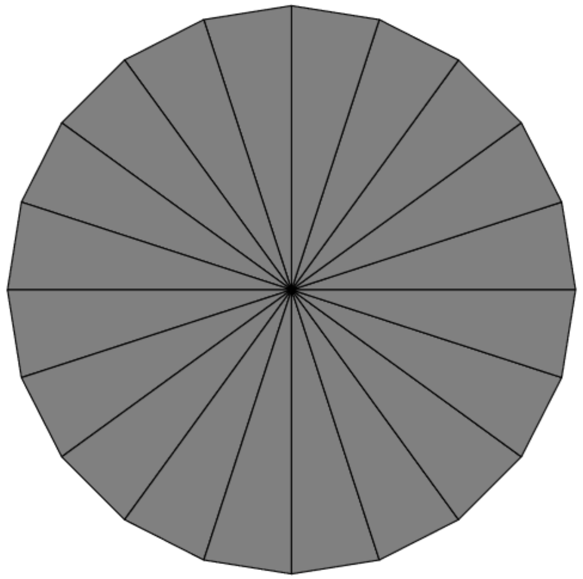




Sparse Matrix Representation

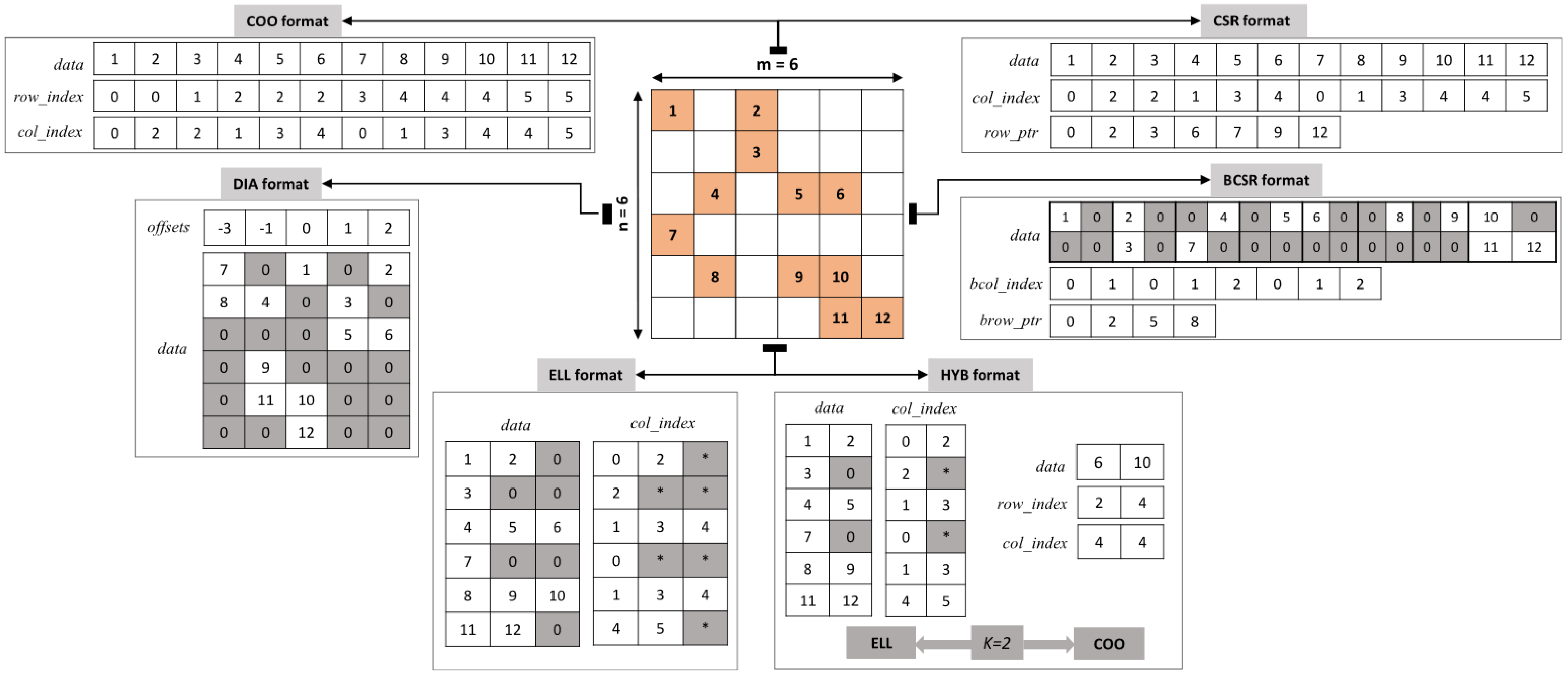
4. Ellpack-Itpack Format (ELL)

- The vertex-edge connectivity of the wheel is not efficiently encoded in ELL format.
- The vast majority of the entries in the data and indices arrays of the ELL representation will be wasted.



2

Sparse Matrix Representation



1		2			
		3			
	4		5	6	
7					
	8		9	10	
				11	12

COO format

data	1	2	3	4	5	6	7	8	9	10	11	12
row_index	0	0	1	2	2	2	3	4	4	4	5	5
col_index	0	2	2	1	3	4	0	1	3	4	4	5

CSR format

data	1	2	3	4	5	6	7	8	9	10	11	12
col_index	0	2	2	1	3	4	0	1	3	4	4	5
row_ptr	0	2	3	6	7	9	12					

DIA format

offsets	-3	-1	0	1	2
data	7	0	1	0	2
	8	4	0	3	0
	0	0	0	5	6
	0	9	0	0	0
	0	11	10	0	0
	0	0	12	0	0

BCSR format

data	1	0	2	0	0	4	0	5	6	0	0	8	0	9	10	0
	0	0	3	0	7	0	0	0	0	0	0	0	0	0	11	12
bcol_index	0	1	0	1	2	0	1	2								
brow_ptr	0	2	5	8												

ELL format

data			*col_index*		
1	2	0	0	2	*
3	0	0	2	*	*
4	5	6	1	3	4
7	0	0	0	*	*
8	9	10	1	3	4
11	12	0	4	5	*

HYB format

data		*col_index*	
1	2	0	2
3	0	2	*
4	5	1	3
7	0	0	*
8	9	1	3
11	12	4	5

data	6	10
row_index	2	4
col_index	4	4

Fig. 1. The sparse matrix example in COO, CSR, BCSR, ELL, DIA, and HYB.

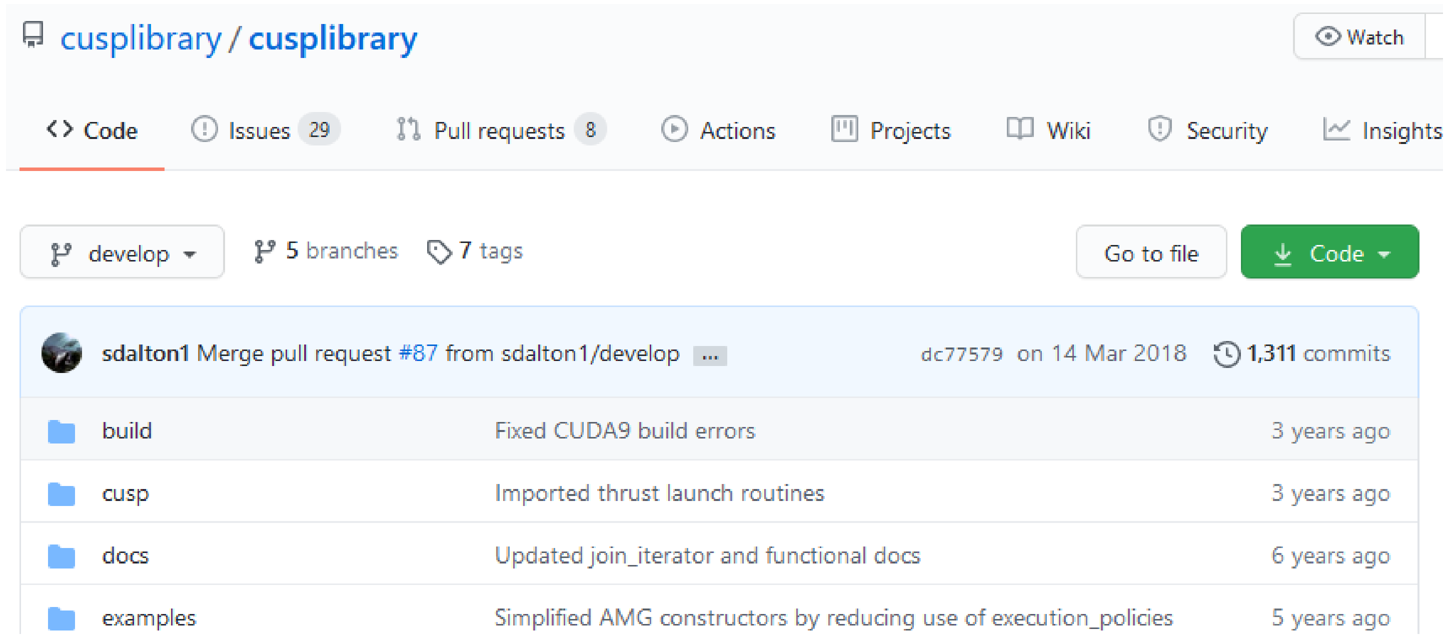
3

Library : CUSP (open source)

http://cusplibrary.github.io/

- Cusp is a library for sparse linear algebra and graph computations based on Thrust.

Current release : v0.5.1 (April 28, 2015)

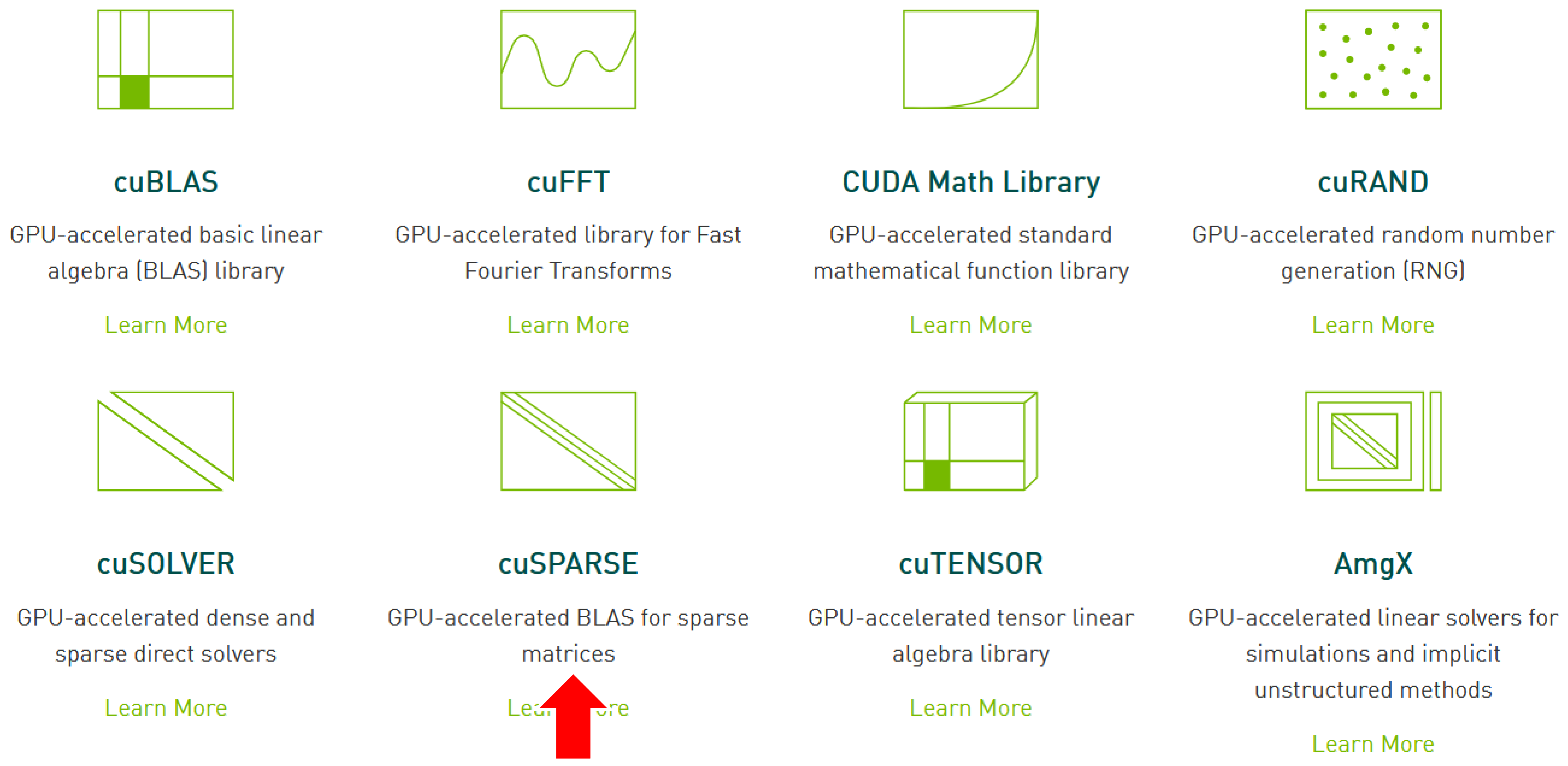


sdalton1 Merge pull request #87 from sdalton1/develop … dc77579 on 14 Mar 2018 1,311 commits

build	Fixed CUDA9 build errors	3 years ago
cusp	Imported thrust launch routines	3 years ago
docs	Updated join_iterator and functional docs	6 years ago
examples	Simplified AMG constructors by reducing use of execution_policies	5 years ago

3 Library : CUDA Toolkit

https://docs.nvidia.com/cuda/pdf/CUSPARSE_Library.pdf



cuBLAS

GPU-accelerated basic linear algebra (BLAS) library

Learn More

cuFFT

GPU-accelerated library for Fast Fourier Transforms

Learn More

CUDA Math Library

GPU-accelerated standard mathematical function library

Learn More

cuRAND

GPU-accelerated random number generation (RNG)

Learn More

cuSOLVER

GPU-accelerated dense and sparse direct solvers

Learn More

cuSPARSE

GPU-accelerated BLAS for sparse matrices

Lea ore

cuTENSOR

GPU-accelerated tensor linear algebra library

Learn More

AmgX

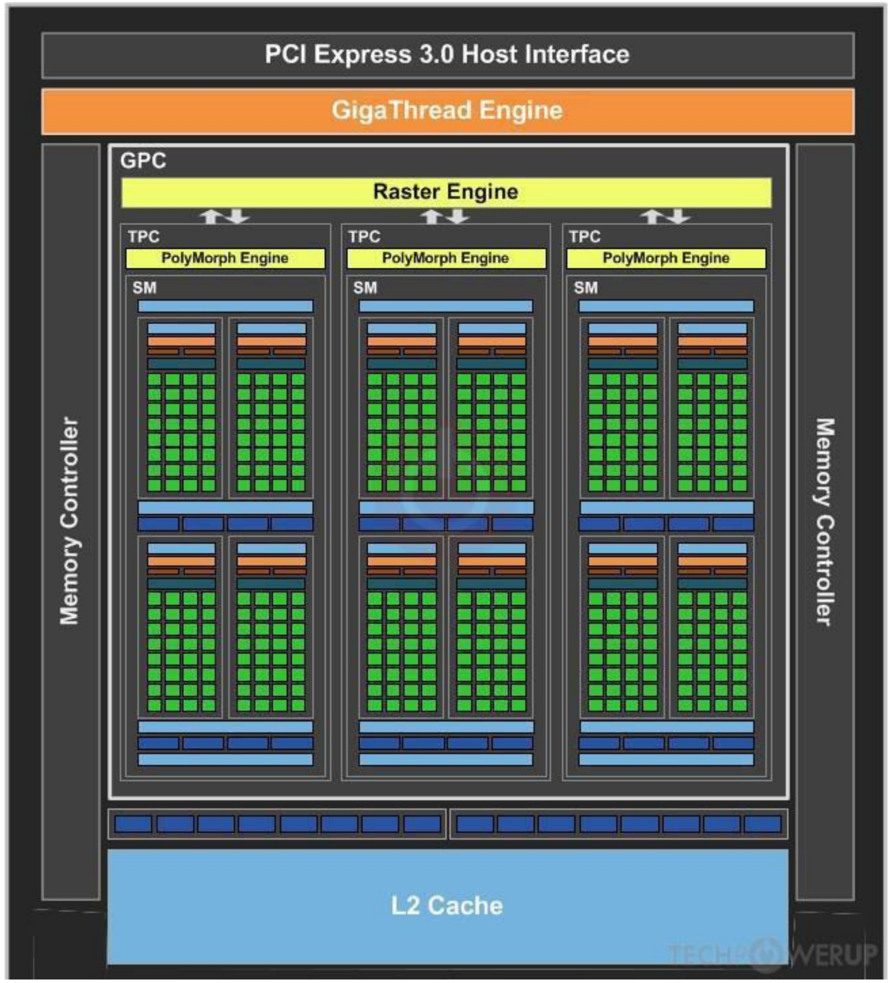
GPU-accelerated linear solvers for simulations and implicit unstructured methods

Learn More

CUDA Implementation

1. Environment

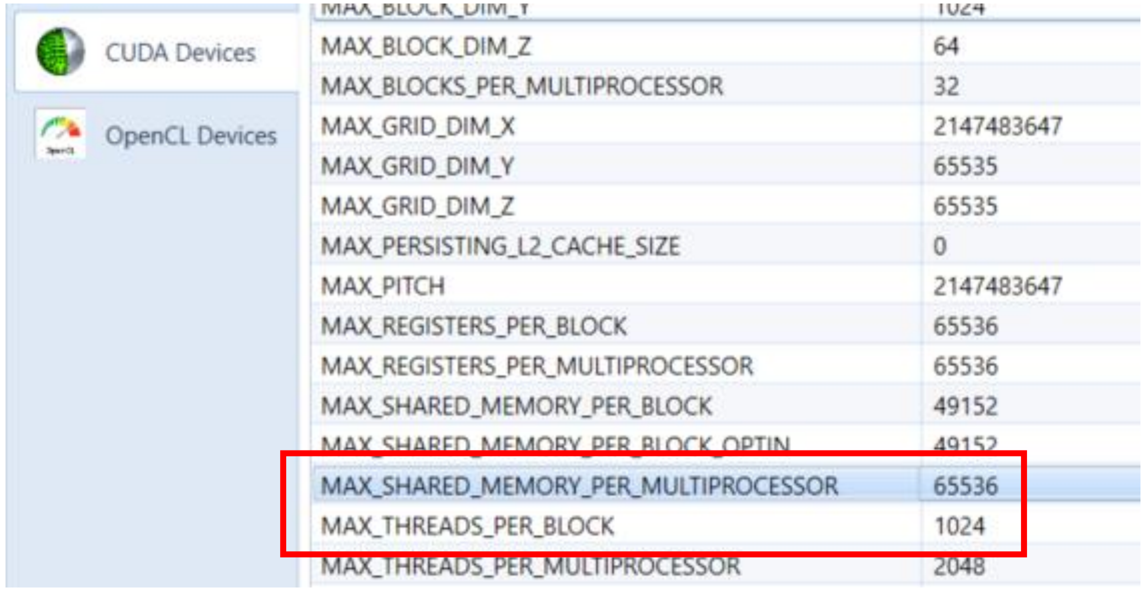
- **GeForce 940M**
- Maxwell Architecture (1st gen) : GM108
- CUDA Compute Compatibility : CUDA 5.0
- 1 GPC
- 3 Maxwell Streaming Multiprocessors
- 128 CUDA core in a single SMM
- 384 CUDA Cores



4

CUDA Implementation

1. Environment

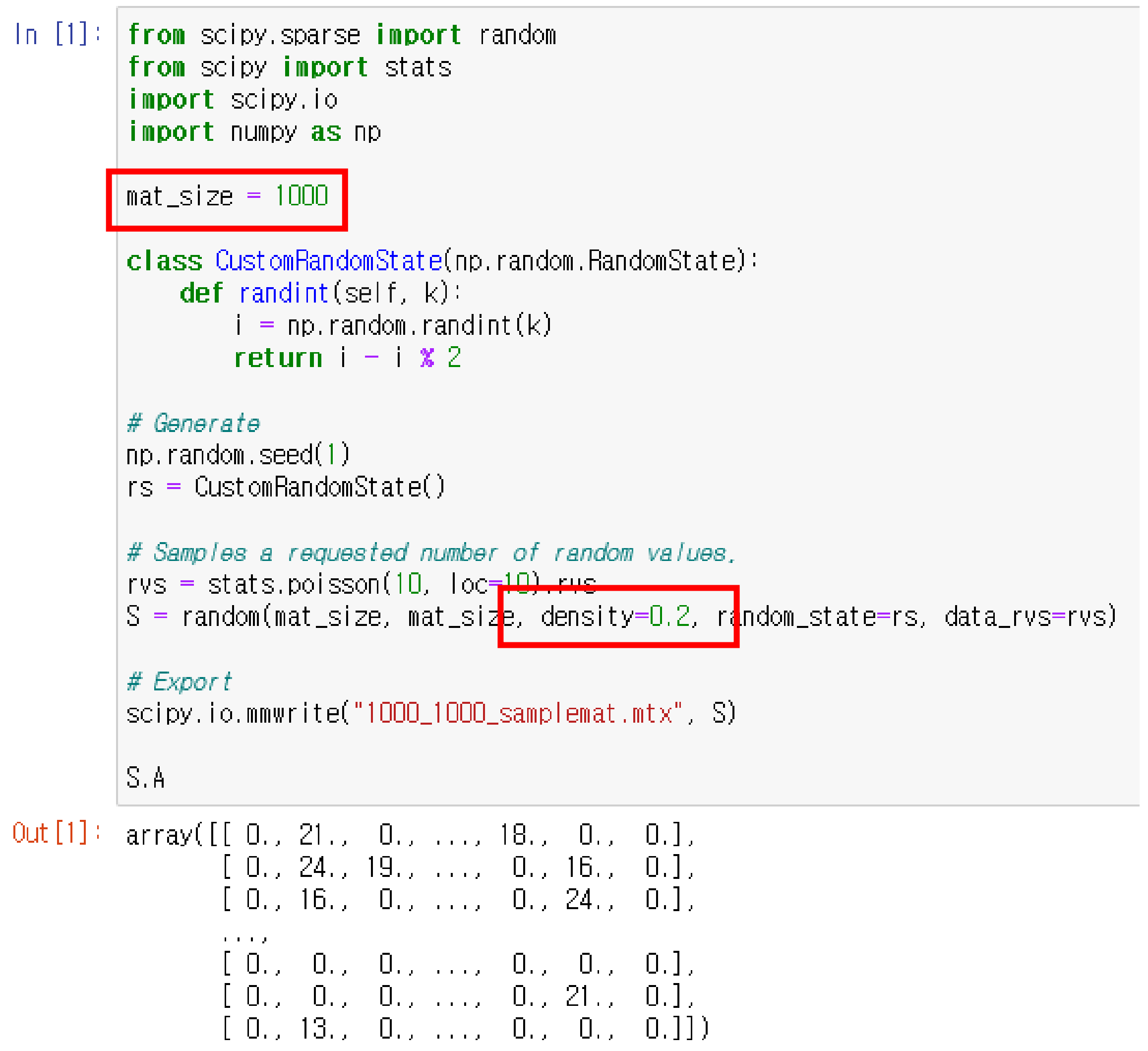


MAX_BLOCK_DIM_Y	1024
MAX_BLOCK_DIM_Z	64
MAX_BLOCKS_PER_MULTIPROCESSOR	32
MAX_GRID_DIM_X	2147483647
MAX_GRID_DIM_Y	65535
MAX_GRID_DIM_Z	65535
MAX_PERSISTING_L2_CACHE_SIZE	0
MAX_PITCH	2147483647
MAX_REGISTERS_PER_BLOCK	65536
MAX_REGISTERS_PER_MULTIPROCESSOR	65536
MAX_SHARED_MEMORY_PER_BLOCK	49152
MAX_SHARED_MEMORY_PER_BLOCK_OPTIN	49152
MAX_SHARED_MEMORY_PER_MULTIPROCESSOR	65536
MAX_THREADS_PER_BLOCK	1024
MAX_THREADS_PER_MULTIPROCESSOR	2048

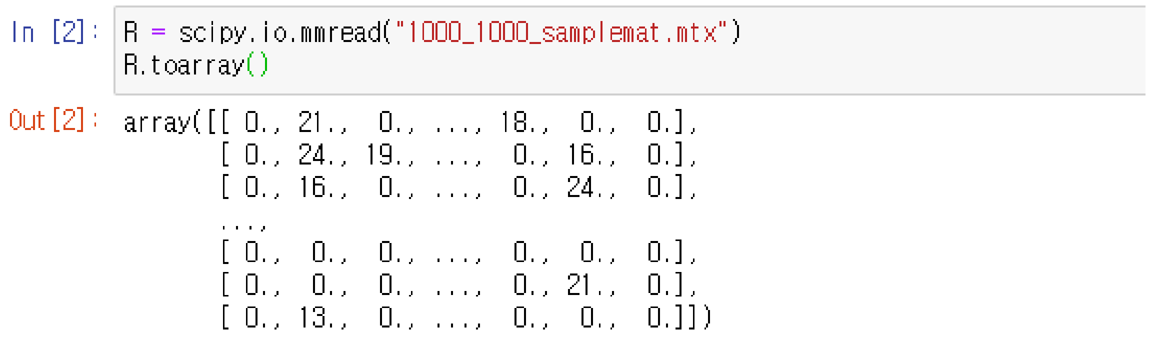
- Max number of concurrent warps/SMM : 64
- Max number of thread blocks/SM : 32
- Register File Size : 64KB
- Shared memory: 64KB

4 CUDA Implementation

2. Matrix Random Generation



```
In [1]: from scipy.sparse import random
        from scipy import stats
        import scipy.io
        import numpy as np

        mat_size = 1000

        class CustomRandomState(np.random.RandomState):
            def randint(self, k):
                i = np.random.randint(k)
                return i - i % 2

        # Generate
        np.random.seed(1)
        rs = CustomRandomState()

        # Samples a requested number of random values.
        rvs = stats.poisson(10, loc=10).rvs
        S = random(mat_size, mat_size, density=0.2, random_state=rs, data_rvs=rvs)

        # Export
        scipy.io.mmwrite("1000_1000_samplemat.mtx", S)

        S.A

Out[1]: array([[ 0., 21.,  0., ..., 18.,  0.,  0.],
               [ 0., 24., 19., ...,  0., 16.,  0.],
               [ 0., 16.,  0., ...,  0., 24.,  0.],
               ...,
               [ 0.,  0.,  0., ...,  0.,  0.,  0.],
               [ 0.,  0.,  0., ...,  0., 21.,  0.],
               [ 0., 13.,  0., ...,  0.,  0.,  0.]])
```

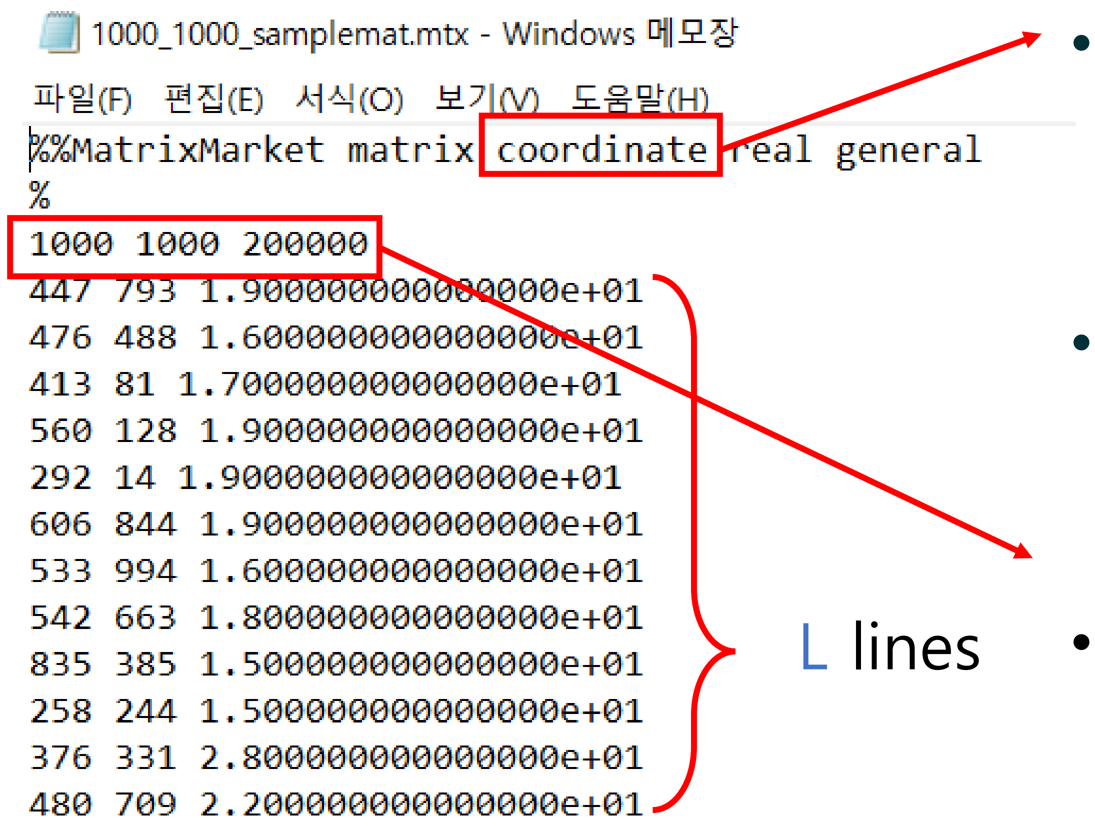


```
In [2]: R = scipy.io.mmread("1000_1000_samplemat.mtx")
        R.toarray()

Out[2]: array([[ 0., 21.,  0., ..., 18.,  0.,  0.],
               [ 0., 24., 19., ...,  0., 16.,  0.],
               [ 0., 16.,  0., ...,  0., 24.,  0.],
               ...,
               [ 0.,  0.,  0., ...,  0.,  0.,  0.],
               [ 0.,  0.,  0., ...,  0., 21.,  0.],
               [ 0., 13.,  0., ...,  0.,  0.,  0.]])
```

https://docs.scipy.org/doc/scipy/reference/sparse.html

4 CUDA Implementation

3. Matrix Market (MM) Exchange Format


1000_1000_samplemat.mtx - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
L lines

```
%%MatrixMarket matrix coordinate real general
%
1000 1000 200000
447 793 1.900000000000000e+01
476 488 1.600000000000000e+01
413 81 1.700000000000000e+01
560 128 1.900000000000000e+01
292 14 1.900000000000000e+01
606 844 1.900000000000000e+01
533 994 1.600000000000000e+01
542 663 1.800000000000000e+01
835 385 1.500000000000000e+01
258 244 1.500000000000000e+01
376 331 2.800000000000000e+01
480 709 2.200000000000000e+01
```

- Coordinate Format : suitable for representing general sparse matrices.
- Only nonzero entries are provided, and the coordinates of each nonzero entry is given explicitly.
- Rows, Columns, entries(L)

https://math.nist.gov/MatrixMarket/formats.html

CUDA Implementation

4. Reading Matrix

http://math.nist.gov/MatrixMarket/mmio/c/mmio.h
http://math.nist.gov/MatrixMarket/mmio/c/mmio.c

```
void read_matrix(int* argJR, int* argJC, double* argAA) {
    int m = M;
    int n = N;
    int nz = NZ;

    FILE* MTX;
    MM_typecode matrix_code;

    MTX = fopen("10_10_sample_mat.mtx", "r");
```

4

CUDA Implementation

4. Reading Matrix

http://math.nist.gov/MatrixMarket/mmio/c/mmio.h
http://math.nist.gov/MatrixMarket/mmio/c/mmio.c

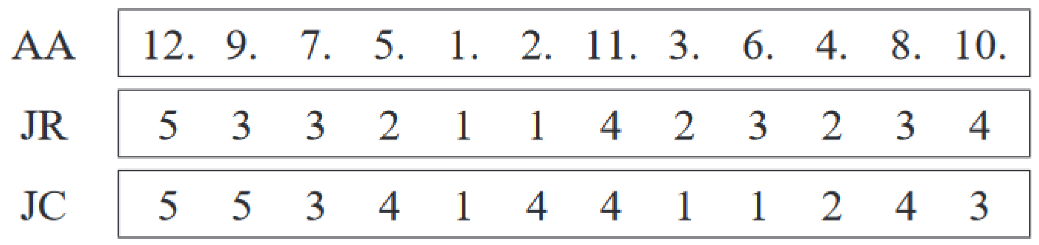
Read banner and size info

```
    // Read banner, type, etc essential infos
    // Verification steps are ignored.
    if (mm_read_banner(MTX, &matrix_code) != 0) exit(1);
    mm_read_mtx_crd_size(MTX, &m, &n, &nz); // Over max 1025

    printf("Market Market type: [%s]\n", mm_typecode_to_str(matrix_code));

    // COO format
    for (register int i = 0; i < NZ; i++)
        fscanf_s(MTX, "%d %d %lf\n", &argJR[i], &argJC[i], &argAA[i]);
    fclose(MTX);
}
```

4 CUDA Implementation

4. Reading Matrix

```c
// ---- main() ----
int main() {
    int* JR = (int*)malloc(NZ * sizeof(int));
    int* JC = (int*)malloc(NZ * sizeof(int));
    double* AA = (double*)malloc(NZ * sizeof(double));

    // prepare elements
    read_matrix(JR, JC, AA);
```

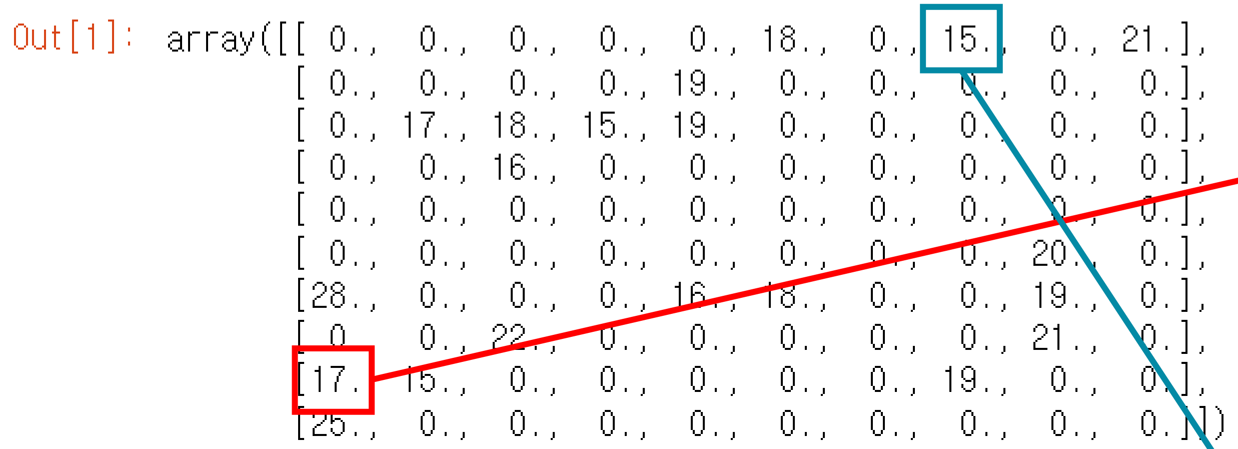


AA	12.	9.	7.	5.	1.	2.	11.	3.	6.	4.	8.	10.
JR	5	3	3	2	1	1	4	2	3	2	3	4
JC	5	5	3	4	1	4	4	1	1	2	4	3

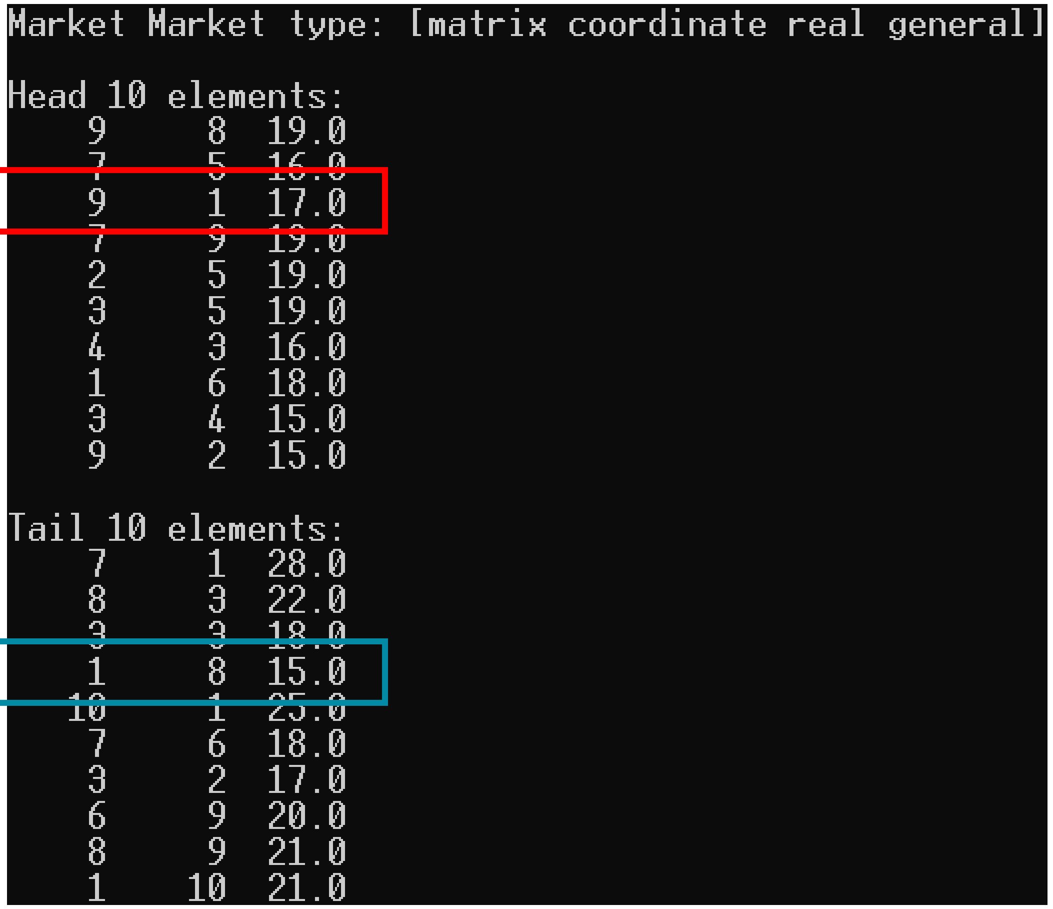
COO Example (Slide 8)

4 CUDA Implementation

4. Reading Matrix



```
Out[1]: array([[ 0.,  0.,  0.,  0.,  0., 18.,  0., 15.,  0., 21.],
               [ 0.,  0.,  0.,  0., 19.,  0.,  0.,  0.,  0.,  0.],
               [ 0., 17., 18., 15., 19.,  0.,  0.,  0.,  0.,  0.],
               [ 0.,  0., 16.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
               [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
               [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., 20.,  0.],
               [28.,  0.,  0.,  0., 16., 18.,  0.,  0., 19.,  0.],
               [ 0.,  0., 22.,  0.,  0.,  0.,  0.,  0., 21.,  0.],
               [17., 15.,  0.,  0.,  0.,  0.,  0., 19.,  0.,  0.],
               [25.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.]])
```



```
Market Market type: [matrix coordinate real general]

Head 10 elements:
    9     8  19.0
    7     5  16.0
    9     1  17.0
    7     9  19.0
    2     5  19.0
    3     5  19.0
    4     3  16.0
    1     6  18.0
    3     4  15.0
    9     2  15.0

Tail 10 elements:
    7     1  28.0
    8     3  22.0
    3     3  18.0
    1     8  15.0
   10     1  25.0
    7     6  18.0
    3     2  17.0
    6     9  20.0
    8     9  21.0
    1    10  21.0
```

- COO Format is not sorted by rows.
- cuSPARSE's data type assumes all elements are row-sorted.
- Order is important for transformation.

CUDA Implementation

5. Implementation

1. Sorting COO format using cuSPARSE
2. COO SpMV using cuSPARSE
3. CSR SpMV using cuSPARSE
4. CSR SpMV using scalar kernel
5. CSR SpMV using grid-stride kernel

4

CUDA Implementation

5.1 Sorting COO format using cuSPARSE

```
//
// ---- Step 2: Handle create, bind a stream ----
int* device_JR = NULL;
int* device_JC = NULL;
double* device_AA = NULL;
double* device_AA_sorted = NULL;
int* device_P = NULL;

void* buffer = NULL;
size_t buffer_size = 0;
```

On device

4 CUDA Implementation

5.1 Sorting COO format using cuSPARSE

Pointer of cuSPARSE context

```
cusparseHandle_t handle = NULL;
cudaStream_t stream = NULL;
```

Creates a new asynchronous stream.
The flags determine the behaviors of the stream.

```
CUDA_ERR(cudaStreamCreateWithFlags(&stream, cudaStreamNonBlocking));
CUSPARSE_ERR(cusparseCreate(&handle));
CUSPARSE_ERR(cusparseSetStream(handle, stream));

CUSPARSE_ERR( // ---- Step 3. Allocate Buffer ----
    cusparseXcoosort_bufferSizeExt(
        handle,
        M, N, NZ,
        device_JR, device_JC, &buffer_size
    )
);
```

Output

CUDA Implementation

5.1 Sorting COO format using cuSPARSE

```
CUDA_ERR(cudaMalloc((void **)&device_JR, sizeof(int) * NZ));
CUDA_ERR(cudaMalloc((void **)&device_JC, sizeof(int) * NZ));
CUDA_ERR(cudaMalloc((void **)&device_P, sizeof(int) * NZ));
CUDA_ERR(cudaMalloc((void **)&device_AA, sizeof(double) * NZ));
CUDA_ERR(cudaMalloc((void **)&device_AA_sorted, sizeof(double) * NZ));
CUDA_ERR(cudaMalloc((void **)&buffer, sizeof(char) * buffer_size));

CUDA_ERR(cudaMemcpy(device_JR, host_JR, sizeof(int) * NZ, cudaMemcpyHostToDevice));
CUDA_ERR(cudaMemcpy(device_JC, host_JC, sizeof(int) * NZ, cudaMemcpyHostToDevice));
CUDA_ERR(cudaMemcpy(device_AA, host_AA, sizeof(double) * NZ, cudaMemcpyHostToDevice));
CUDA_ERR(cudaDeviceSynchronize());
```

4 CUDA Implementation

5.1 Sorting COO format using cuSPARSE

```
//
// ---- Step 4. Setup permutation vector P to Identity ----
CUSPARSE_ERR(cusparseCreateIdentityPermutation(handle, NZ, device_P));

// ---- Step 5. Sort ----
CUSPARSE_ERR(
        cusparseXcoosortByRow(handle, M, N, NZ, device_JR, device_JC, device_P, buffer)
);

// Gather
CUSPARSE_ERR(cusparseDgthr(
        handle, NZ, device_AA, device_AA_sorted, device_P, CUSPARSE_INDEX_BASE_ZERO));
CUDA_ERR(cudaDeviceSynchronize());
```

Gathers the elements of the vector listed in the index array into the second vector.

CUDA Implementation

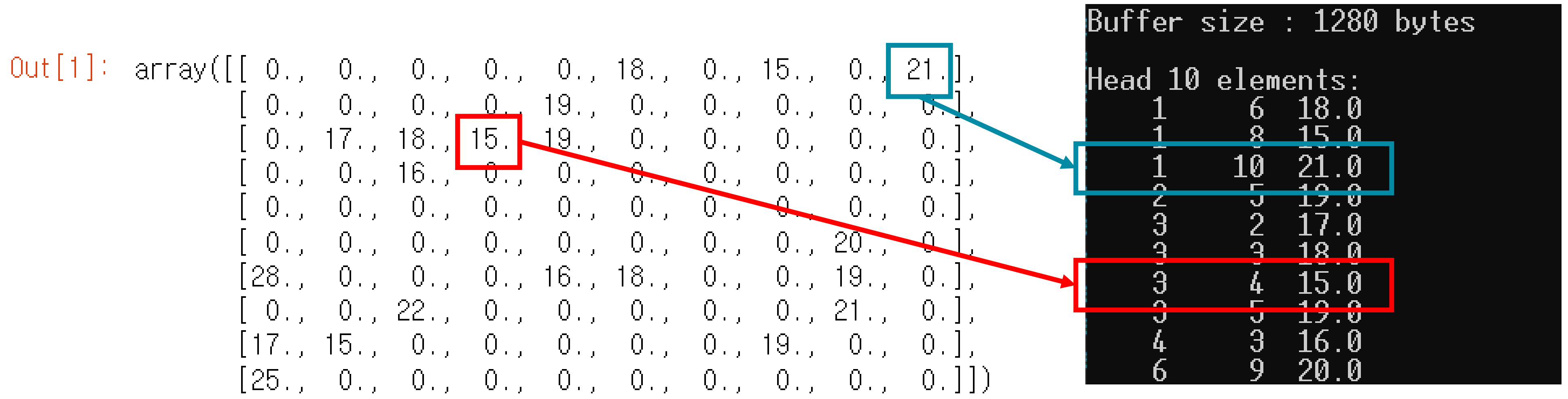
5.1 Sorting COO format using cuSPARSE

```
// Fetch back
CUDA_ERR(cudaMemcpy(host_JR, device_JR, sizeof(int) * NZ, cudaMemcpyDeviceToHost));
CUDA_ERR(cudaMemcpy(host_JC, device_JC, sizeof(int) * NZ, cudaMemcpyDeviceToHost));
CUDA_ERR(cudaMemcpy(host_P, device_P, sizeof(int) * NZ, cudaMemcpyDeviceToHost));
CUDA_ERR(cudaMemcpy(host_AA, device_AA_sorted, sizeof(double) * NZ,
cudaMemcpyDeviceToHost));
CUDA_ERR(cudaDeviceSynchronize());
```

- No __**global**__ syntax.
- cuSPARSE API controls everything.

CUDA Implementation

5.1 Sorting COO format using cuSPARSE



```
Out[1]: array([[ 0.,  0.,  0.,  0.,  0., 18.,  0., 15.,  0., 21.],
               [ 0.,  0.,  0.,  0., 19.,  0.,  0.,  0.,  0.,  0.],
               [ 0., 17., 18., 15., 19.,  0.,  0.,  0.,  0.,  0.],
               [ 0.,  0., 16.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
               [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
               [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., 20.,  0.],
               [28.,  0.,  0.,  0., 16., 18.,  0.,  0., 19.,  0.],
               [ 0.,  0., 22.,  0.,  0.,  0.,  0.,  0., 21.,  0.],
               [17., 15.,  0.,  0.,  0.,  0.,  0., 19.,  0.,  0.],
               [25.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.]])
```

```
Buffer size : 1280 bytes

Head 10 elements:
    1      6   18.0
    1      8   15.0
    1     10   21.0
    2      5   19.0
    3      2   17.0
    3      3   18.0
    3      4   15.0
    3      5   19.0
    4      3   16.0
    6      9   20.0
```

- Elements are sorted by row nicely!

4

CUDA Implementation

5.2 COO SpMV using cuSPARSE

```
// ---- Step 7. Define variables
const float alpha = 1;
const float beta = 0;

float host_y[N] = {0, };
float host_x[M];

float* device_x = NULL;
float* device_y = NULL;

for (auto& elem : host_x) elem = 1;
```

$$\boxed{y} = A\boxed{x}$$

4

CUDA Implementation

5.2 COO SpMV using cuSPARSE

Sparse Matrix Descriptor Type

Dense Vector Descriptor Type

```
cusparseSpMatDescr_t sp_mtx; // device
cusparseDnVecDescr_t dn_x, dn_y; // device

// ---- Step 8. Get your GPU memory ready ----
CUDA_ERR(cudaMalloc((void**)&device_x, sizeof(float) * M));
CUDA_ERR(cudaMalloc((void**)&device_y, sizeof(float) * N));

CUDA_ERR(cudaMemcpy(device_x, host_x, sizeof(float) * M, cudaMemcpyHostToDevice));
CUDA_ERR(cudaMemcpy(device_y, host_y, sizeof(float) * N, cudaMemcpyHostToDevice));

CUSPARSE_ERR(cusparseCreate(&handle));
```

4 CUDA Implementation

5.2 COO SpMV using cuSPARSE

```
// Create sparse matrix in COO format
CUSPARSE_ERR(
    cusparseCreateCoo(
        &sp_mtx,
        M, N, NZ, device_JR, device_JC, device_AA_sorted,
        CUSPARSE_INDEX_32I, CUSPARSE_INDEX_BASE_ONE, CUDA_R_32F)
);

CUSPARSE_ERR(cusparseCreateDnVec(&dn_x, N, device_x, CUDA_R_32F));
CUSPARSE_ERR(cusparseCreateDnVec(&dn_y, M, device_y, CUDA_R_32F));
```

Real 32 Float : Precision

Index starting point

CUDA Implementation

5.2 COO SpMV using cuSPARSE

```
CUSPARSE_ERR(cusparseSpMV_bufferSize(
    handle, CUSPARSE_OPERATION_NON_TRANSPOSE,
    &alpha, sp_mtx, dn_x, &beta, dn_y, CUDA_R_32F,
    CUSPARSE_COOMV_ALG, &buffer_size));

printf("Buffer size : %lld bytes \n", (long long)buffer_size);
CUDA_ERR(cudaMalloc(&buffer, buffer_size));
```

4 CUDA Implementation

5.2 COO SpMV using cuSPARSE

This function performs the multiplication of a sparse matrix `matA` and a dense vector `vecX`

$$\mathbf{Y} = \alpha op(\mathbf{A}) \cdot \mathbf{X} + \beta \mathbf{Y}$$

where

- `op(A)` is a sparse matrix of size $m \times k$
- `X` is a dense vector of size k
- `Y` is a dense vector of size m
- α and β are scalars

Also, for matrix `A`

$$\text{op}(A) = \begin{cases} A & \text{if op(A) == } \boxed{\text{CUSPARSE\_ OPERATION\_NON\_TRANSPOSE}} \\ A^T & \text{if op(A) == CUSPARSE\_ OPERATION\_TRANSPOSE} \\ A^H & \text{if op(A) == CUSPARSE\_ OPERATION\_CONJUGATE\_TRANSPOSE} \end{cases}$$

CUDA Implementation

5.2 COO SpMV using cuSPARSE

```
// ---- Step 9. Do SpMV ----
CUSPARSE_ERR(cusparseSpMV(handle, CUSPARSE_OPERATION_NON_TRANSPOSE,
    &alpha, sp_mtx, dn_x, &beta, dn_y, CUDA_R_32F,
    CUSPARSE_COOMV_ALG, buffer));
```

Algorithm selection

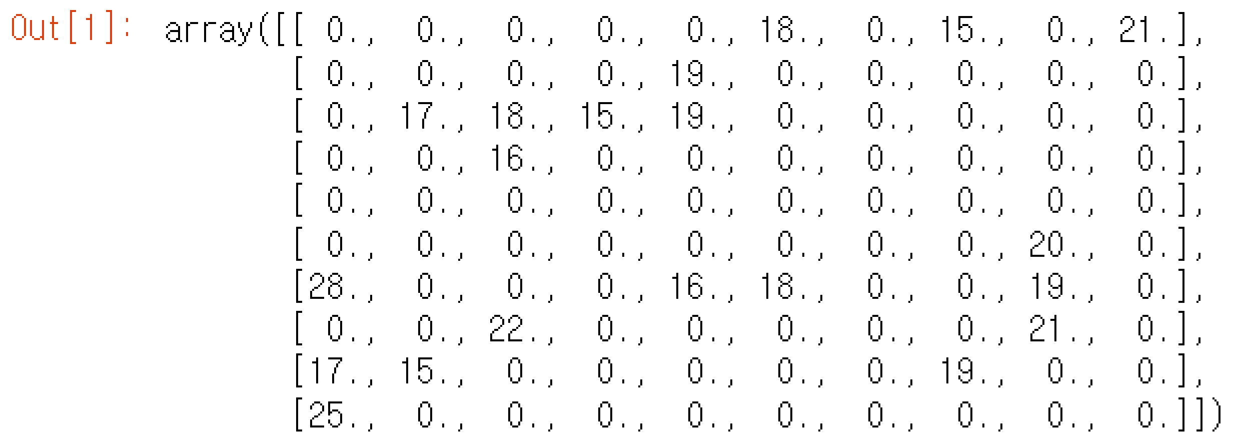
A/X, Y, ComputeType must be matched:

e.g. Uniform-precision: Set everything to CUDA_R_32F

```
// ---- Step 10. Fetch the result ----
CUDA_ERR(cudaMemcpy(host_y, device_y, N * sizeof(float), cudaMemcpyDeviceToHost));
```

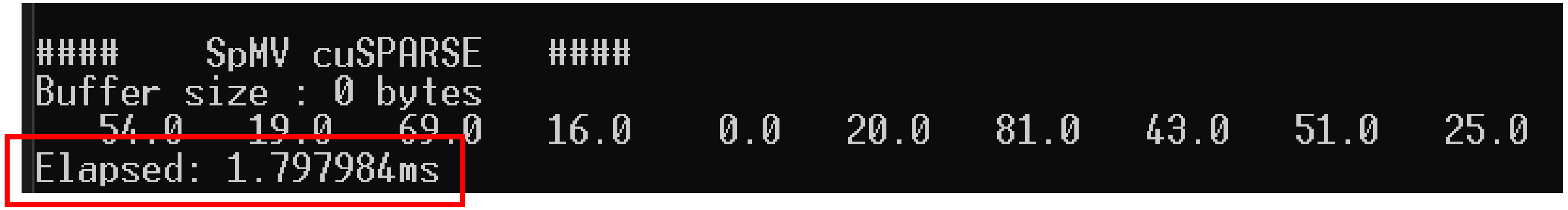
4 CUDA Implementation

5.2 COO SpMV using cuSPARSE



```
Out[1]: array([[ 0.,  0.,  0.,  0.,  0., 18.,  0., 15.,  0., 21.],
               [ 0.,  0.,  0.,  0., 19.,  0.,  0.,  0.,  0.,  0.],
               [ 0., 17., 18., 15., 19.,  0.,  0.,  0.,  0.,  0.],
               [ 0.,  0., 16.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
               [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
               [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., 20.,  0.],
               [28.,  0.,  0.,  0., 16., 18.,  0.,  0., 19.,  0.],
               [ 0.,  0., 22.,  0.,  0.,  0.,  0.,  0., 21.,  0.],
               [17., 15.,  0.,  0.,  0.,  0.,  0., 19.,  0.,  0.],
               [25.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.]])
```

$$X = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \qquad Y = \begin{bmatrix} 18+15+21 \\ 19 \\ 17+18+15+19 \\ 16 \\ 0 \\ 20 \\ 28+16+18+19 \\ 22+21 \\ 17+15+19 \\ 25 \end{bmatrix}$$

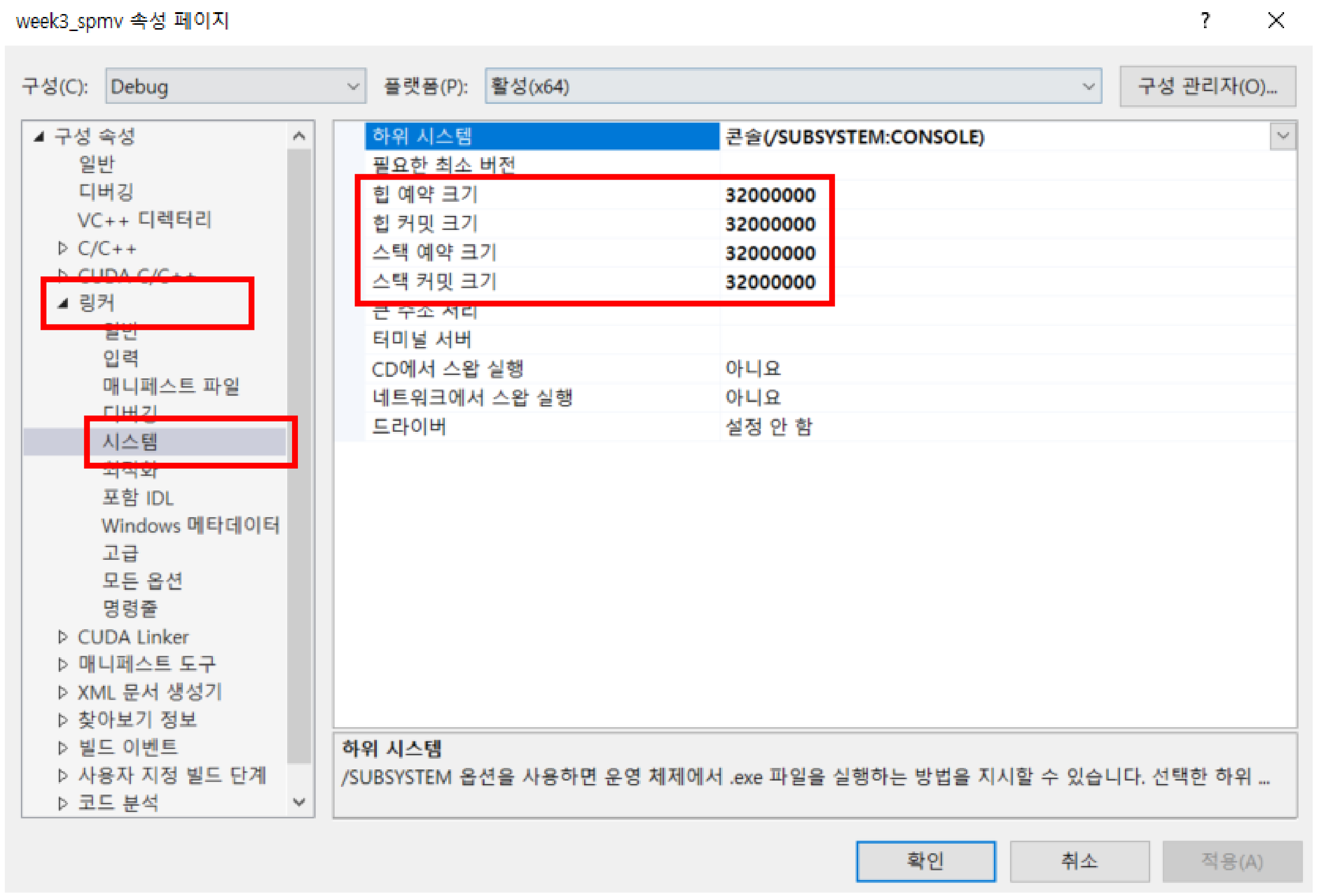


```
####     SpMV cuSPARSE    ####
Buffer size : 0 bytes
   54.0    19.0    69.0    16.0     0.0    20.0    81.0    43.0    51.0    25.0
Elapsed: 1.797984ms
```

- Time recorded only at SpMV calculation step.

CUDA Implementation

5.2 COO SpMV using cuSPARSE



week3_spmv 속성 페이지

구성(C): Debug 플랫폼(P): 활성(x64) 구성 관리자(O)...

- 구성 속성
  - 일반
  - 디버깅
  - VC++ 디렉터리
  - C/C++
  - CUDA C/C++
  - 링커
    - 일반
    - 입력
    - 매니페스트 파일
    - 디버깅
    - 시스템
    - 최적화
    - 포함 IDL
    - Windows 메타데이터
    - 고급
    - 모든 옵션
    - 명령줄
  - CUDA Linker
  - 매니페스트 도구
  - XML 문서 생성기
  - 찾아보기 정보
  - 빌드 이벤트
  - 사용자 지정 빌드 단계
  - 코드 분석

하위 시스템	콘솔(/SUBSYSTEM:CONSOLE)
필요한 최소 버전	
힙 예약 크기	32000000
힙 커밋 크기	32000000
스택 예약 크기	32000000
스택 커밋 크기	32000000
큰 주소 처리	
터미널 서버	
CD에서 스왑 실행	아니요
네트워크에서 스왑 실행	아니요
드라이버	설정 안 함

하위 시스템
/SUBSYSTEM 옵션을 사용하면 운영 체제에서 .exe 파일을 실행하는 방법을 지시할 수 있습니다. 선택한 하위 ...

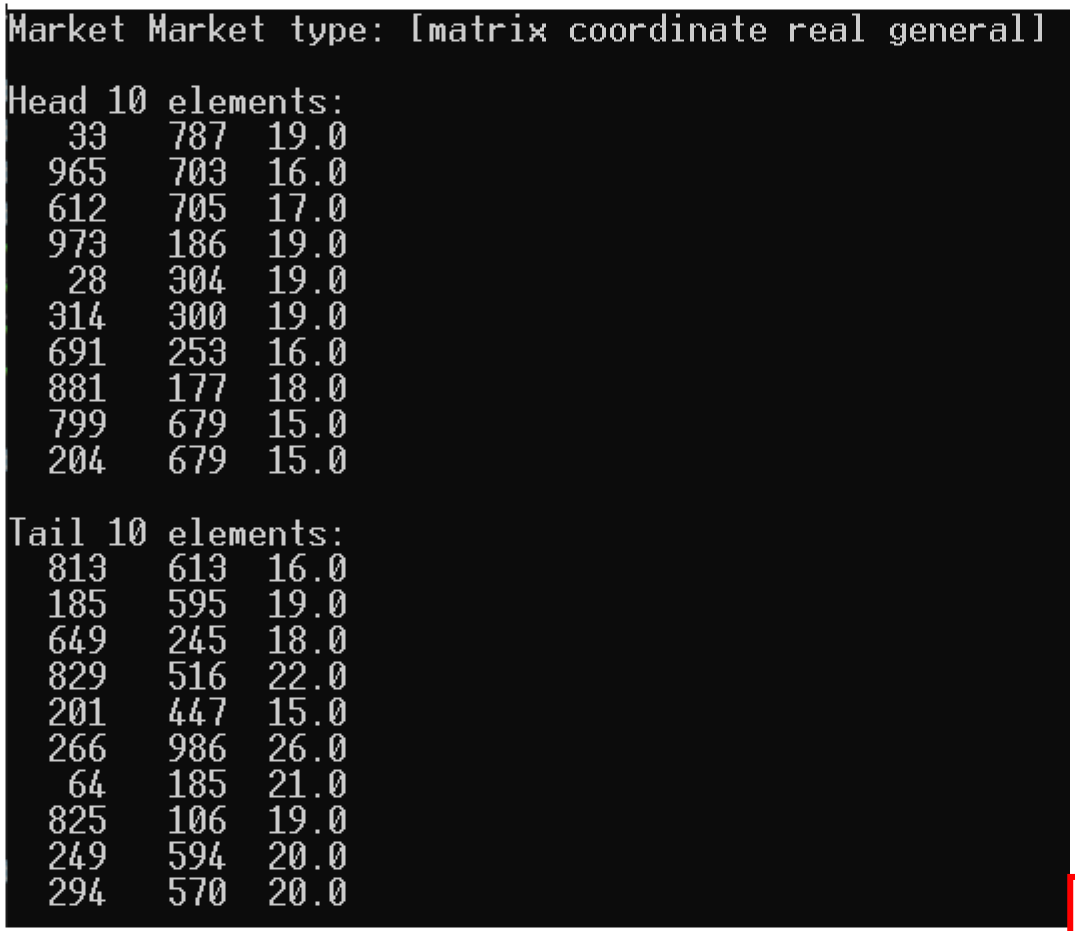
확인 취소 적용(A)

- Matrix 1024 by 1024
- Need to increase stack and heap size
- Default stack : 1MB
- Default heap : 1MB

- Otherwise, stack overflow problem occurs.

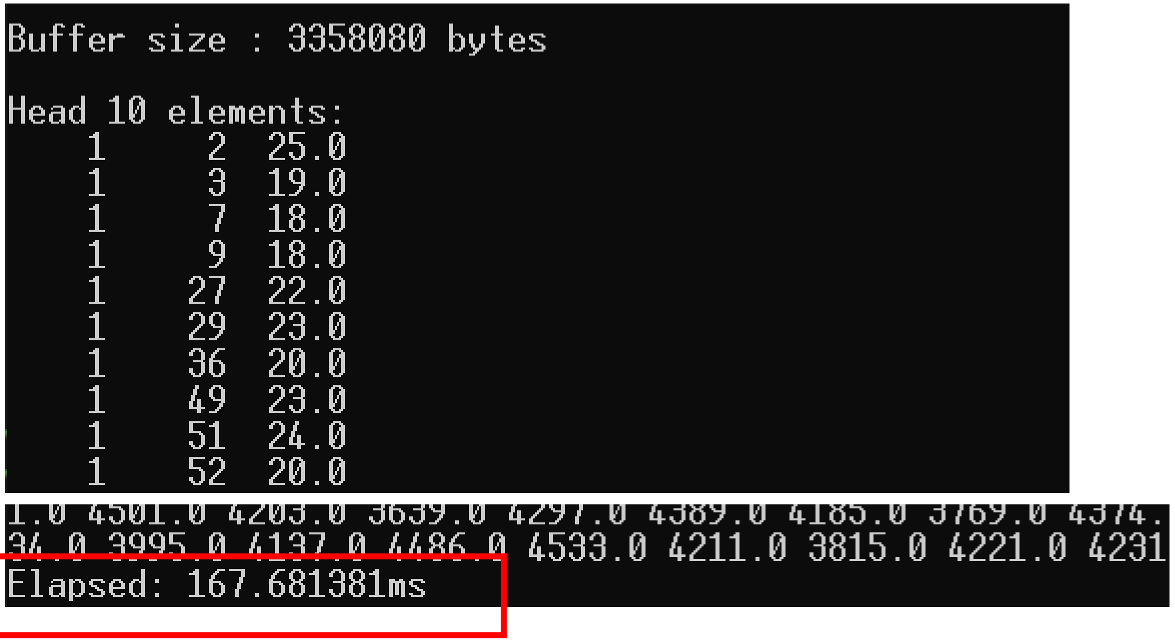
4 CUDA Implementation

5.2 COO SpMV using cuSPARSE



```
Market Market type: [matrix coordinate real general]

Head 10 elements:
   33    787   19.0
  965    703   16.0
  612    705   17.0
  973    186   19.0
   28    304   19.0
  314    300   19.0
  691    253   16.0
  881    177   18.0
  799    679   15.0
  204    679   15.0

Tail 10 elements:
  813    613   16.0
  185    595   19.0
  649    245   18.0
  829    516   22.0
  201    447   15.0
  266    986   26.0
   64    185   21.0
  825    106   19.0
  249    594   20.0
  294    570   20.0
```

```
#define MTX_FILE "1024_1024_sample_mat.mtx“
#define M        1024
#define N        1024
#define NZ       209715
```



```
Buffer size : 3358080 bytes

Head 10 elements:
    1      2   25.0
    1      3   19.0
    1      7   18.0
    1      9   18.0
    1     27   22.0
    1     29   23.0
    1     36   20.0
    1     49   23.0
    1     51   24.0
    1     52   20.0
```

```
1.0 4501.0 4203.0 3639.0 4297.0 4389.0 4185.0 3769.0 4374.
34.0 3995.0 4137.0 4486.0 4533.0 4211.0 3815.0 4221.0 4231
Elapsed: 167.681381ms
```

CUDA Implementation

5.3 CSR SpMV using cuSPARSE

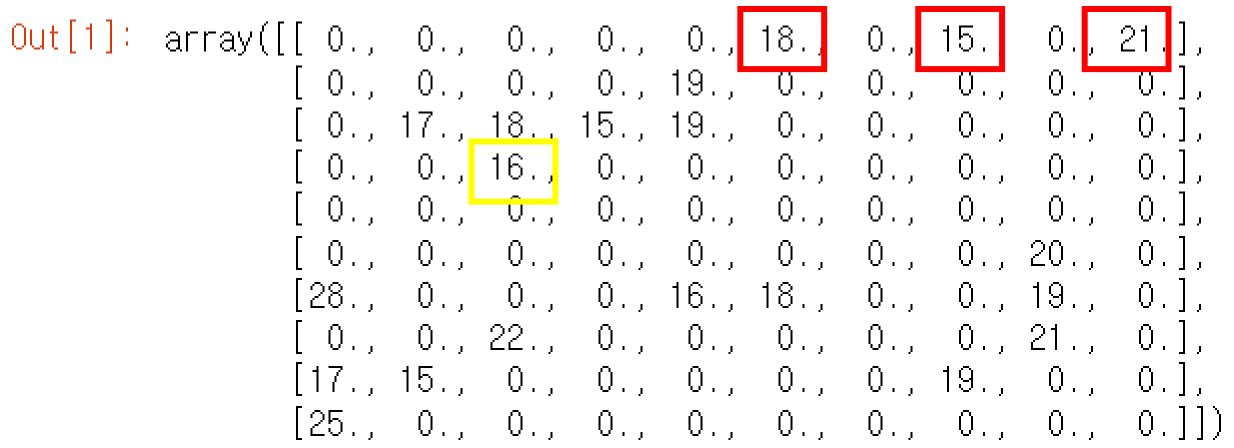
```
#if defined( CUSPARSE_CSR )
    int* t_JR = (int*)calloc((M + 1), sizeof(int));
    int* t_JC = (int*)malloc(NZ * sizeof(int));
    float* t_AA = (float*)malloc(NZ * sizeof(float));
    for (int i = 0; i < M + 1; i++) t_JR[i]++;

    for (int i = 0; i < NZ; i++) {
        t_AA[i] = host_AA[i];
        t_JC[i] = host_JC[i];
        t_JR[host_JR[i]]++;  Count
    }

    for (int i = 0; i < M; i++) t_JR[i + 1] += (t_JR[i] - 1);  Accumulate
```

- Add below Step 6.
- Transforms COO into CSR format.

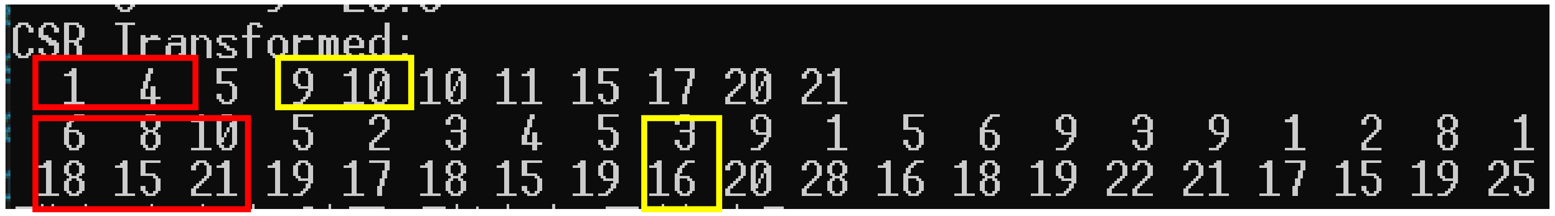
4 CUDA Implementation

5.3 CSR SpMV using cuSPARSE

```
    free(host_JR);
    free(host_JC);
    free(host_AA);

    host_JR = t_JR;
    host_JC = t_JC;
    host_AA = t_AA;
#endif
```



```
Out[1]: array([[ 0.,  0.,  0.,  0.,  0., 18.,  0., 15.,  0., 21.],
               [ 0.,  0.,  0.,  0., 19.,  0.,  0.,  0.,  0.,  0.],
               [ 0., 17., 18., 15., 19.,  0.,  0.,  0.,  0.,  0.],
               [ 0.,  0., 16.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
               [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
               [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., 20.,  0.],
               [28.,  0.,  0.,  0., 16., 18.,  0.,  0., 19.,  0.],
               [ 0.,  0., 22.,  0.,  0.,  0.,  0.,  0., 21.,  0.],
               [17., 15.,  0.,  0.,  0.,  0.,  0., 19.,  0.,  0.],
               [25.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.]])
```



```
CSR Transformed:
  1  4  5  9 10 10 11 15 17 20 21
  6  8 10  5  2  3  4  5  3  9  1  5  6  9  3  9  1  2  8  1
 18 15 21 19 17 18 15 19 16 20 28 16 18 19 22 21 17 15 19 25
```

4 CUDA Implementation

5.3 CSR SpMV using cuSPARSE

Reduced memory space

```
#ifdef CUSPARSE_CSR
    CUDA_ERR(cudaMalloc((void**)&device_JR, sizeof(int) * (M + 1)));
    CUDA_ERR(cudaMalloc((void**)&device_JC, sizeof(int) * NZ));

    CUDA_ERR(cudaMemcpy(device_JR, host_JR, sizeof(int) * (M + 1),
    cudaMemcpyHostToDevice));
    CUDA_ERR(cudaMemcpy(device_JC, host_JC, sizeof(int) * NZ,
    cudaMemcpyHostToDevice));
#endif
```

- Add below Step 8.
- Additional communication between GPU and main memory.

CUDA Implementation

5.3 CSR SpMV using cuSPARSE

```
#ifndef CUSPARSE_CSR
    CUSPARSE_ERR(
        cusparseCreateCoo(
            &sp_mtx,
            M, N, NZ, device_JR, device_JC, device_AA_sorted,
            CUSPARSE_INDEX_32I, CUSPARSE_INDEX_BASE_ONE, CUDA_R_32F)
    );
#else
    CUSPARSE_ERR(
        cusparseCreateCsr(
            &sp_mtx,
            M, N, NZ, device_JR, device_JC, device_AA_sorted,
            CUSPARSE_INDEX_32I, CUSPARSE_INDEX_32I, CUSPARSE_INDEX_BASE_ONE, CUDA_R_32F)
    );
#endif
```

CUDA Implementation

5.3 CSR SpMV using cuSPARSE

```
#ifndef CUSPARSE_CSR
    CUSPARSE_ERR(cusparseSpMV_bufferSize(
        handle, CUSPARSE_OPERATION_NON_TRANSPOSE,
        &alpha, sp_mtx, dn_x, &beta, dn_y, CUDA_R_32F,
        CUSPARSE_COOMV_ALG, &buffer_size));
#else
    CUSPARSE_ERR(cusparseSpMV_bufferSize(
        handle, CUSPARSE_OPERATION_NON_TRANSPOSE,
        &alpha, sp_mtx, dn_x, &beta, dn_y, CUDA_R_32F,
        CUSPARSE_CSRMV_ALG1, &buffer_size));
#endif
```

CUDA Implementation

5.3 CSR SpMV using cuSPARSE

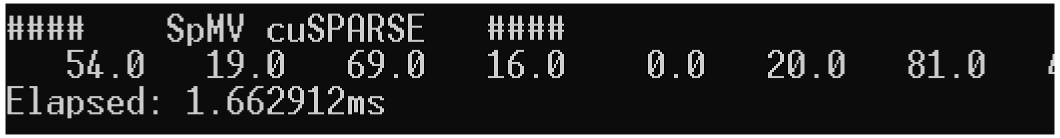
```
#ifndef CUSPARSE_CSR
    CUSPARSE_ERR(cusparseSpMV(handle, CUSPARSE_OPERATION_NON_TRANSPOSE,
        &alpha, sp_mtx, dn_x, &beta, dn_y, CUDA_R_32F,
        CUSPARSE_COOMV_ALG, buffer));
#else
    CUSPARSE_ERR(cusparseSpMV(handle, CUSPARSE_OPERATION_NON_TRANSPOSE,
        &alpha, sp_mtx, dn_x, &beta, dn_y, CUDA_R_32F,
        CUSPARSE_CSRMV_ALG1, buffer));
#endif
```

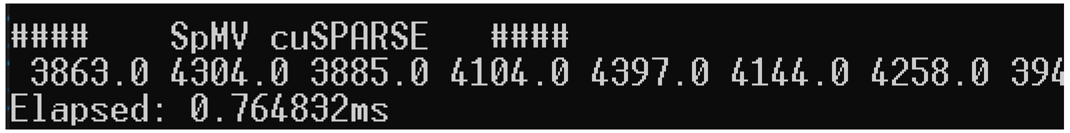
CUDA Implementation

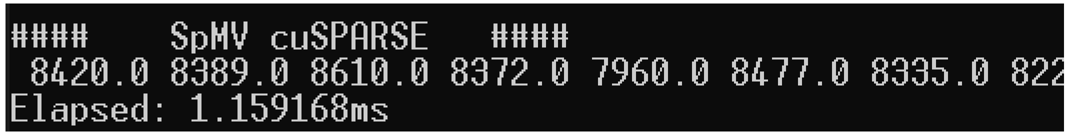
5.3 CSR SpMV using cuSPARSE

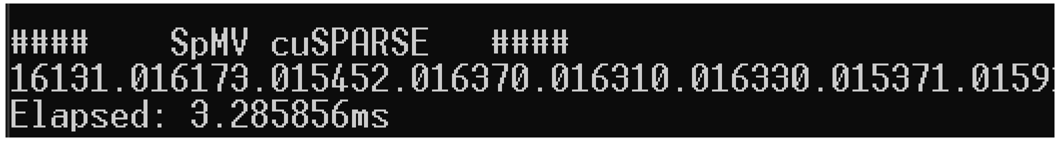
- COO Format

Size	NZ	Results

10×10	20	 `####    SpMV cuSPARSE    ####`
		`  54.0    19.0    69.0    16.0     0.0    20.0    81.0`
		`Elapsed: 1.662912ms`

$1{,}024 \times 1{,}024$	209,715	 `####    SpMV cuSPARSE    ####`
		` 3863.0 4304.0 3885.0 4104.0 4397.0 4144.0 4258.0 39`
		`Elapsed: 0.764832ms`

$2{,}048 \times 2{,}048$	838,861	 `####    SpMV cuSPARSE    ####`
		` 8420.0 8389.0 8610.0 8372.0 7960.0 8477.0 8335.0 822`
		`Elapsed: 1.159168ms`

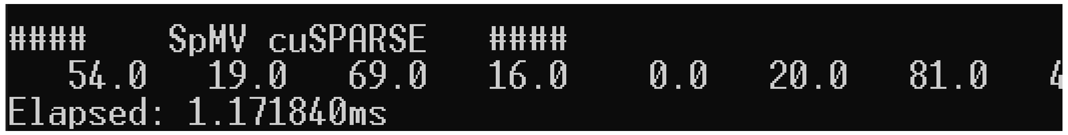
$4{,}096 \times 4{,}096$	3,355,443	 `####    SpMV cuSPARSE    ####`
		`16131.016173.015452.016370.016310.016330.015371.0159`
		`Elapsed: 3.285856ms`

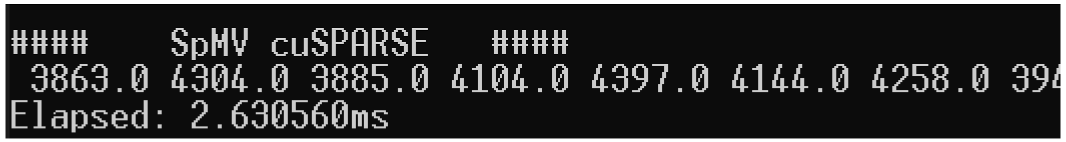
4 CUDA Implementation

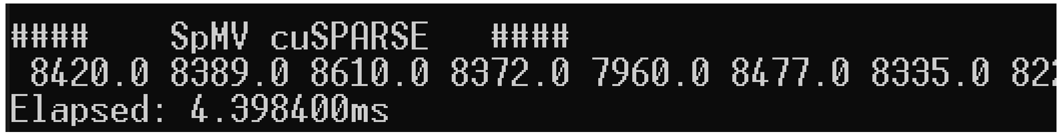
5.3 CSR SpMV using cuSPARSE

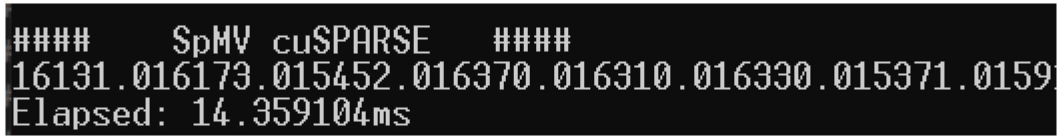
- CSR Format

Size	NZ	Results

10×10	20	 `####     SpMV cuSPARSE    ####`
		`   54.0    19.0    69.0    16.0     0.0    20.0    81.0    4`
		`Elapsed: 1.171840ms`

$1,024 \times 1,024$	209,715	 `####     SpMV cuSPARSE    ####`
		` 3863.0 4304.0 3885.0 4104.0 4397.0 4144.0 4258.0 394`
		`Elapsed: 2.630560ms`

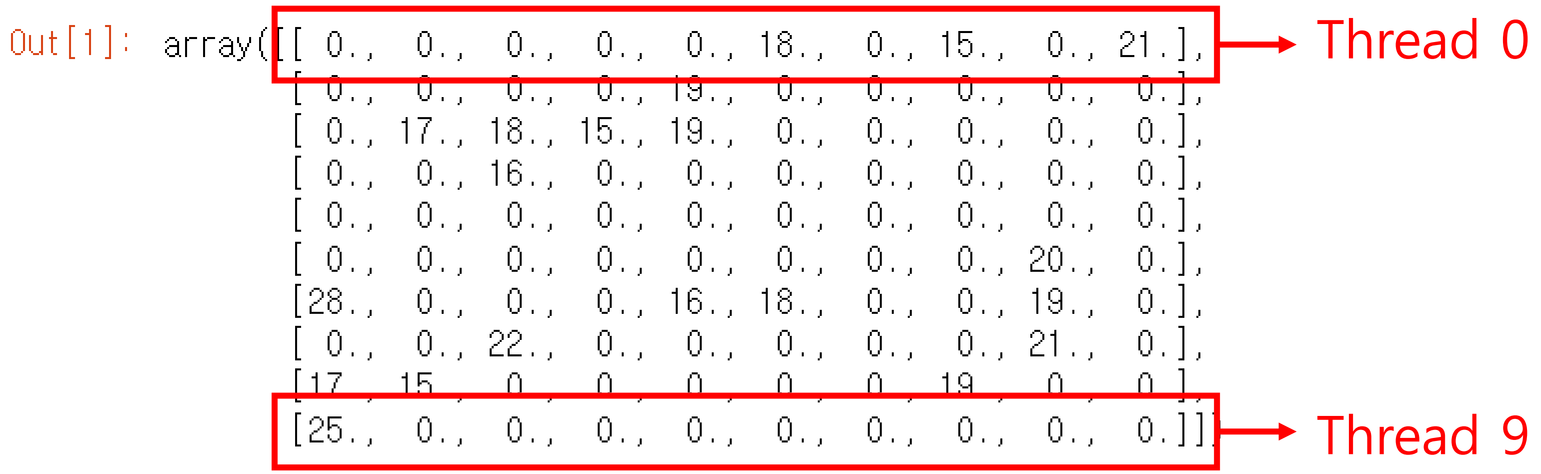
$2,048 \times 2,048$	838,861	 `####     SpMV cuSPARSE    ####`
		` 8420.0 8389.0 8610.0 8372.0 7960.0 8477.0 8335.0 82`
		`Elapsed: 4.398400ms`

$4,096 \times 4,096$	3,355,443	 `####     SpMV cuSPARSE    ####`
		`16131.016173.015452.016370.016310.016330.015371.0159`
		`Elapsed: 14.359104ms`

4 CUDA Implementation

5.4 CSR SpMV using Kernel

- Straightforward method: One thread per row.



```
Out[1]: array([[ 0.,  0.,  0.,  0.,  0., 18.,  0., 15.,  0., 21.],
               [ 0.,  0.,  0.,  0., 19.,  0.,  0.,  0.,  0.,  0.],
               [ 0., 17., 18., 15., 19.,  0.,  0.,  0.,  0.,  0.],
               [ 0.,  0., 16.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
               [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
               [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., 20.,  0.],
               [28.,  0.,  0.,  0., 16., 18.,  0.,  0., 19.,  0.],
               [ 0.,  0., 22.,  0.,  0.,  0.,  0.,  0., 21.,  0.],
               [17., 15.,  0.,  0.,  0.,  0.,  0., 19.,  0.,  0.],
               [25.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.]])
```

Steinberger, M., Derlery, A., Zayer, R., & Seidel, H. P. (2016, September). How naive is naive SpMV on the GPU?. In *2016 IEEE High Performance Extreme Computing Conference (HPEC)* (pp. 1-8). IEEE.

4

CUDA Implementation

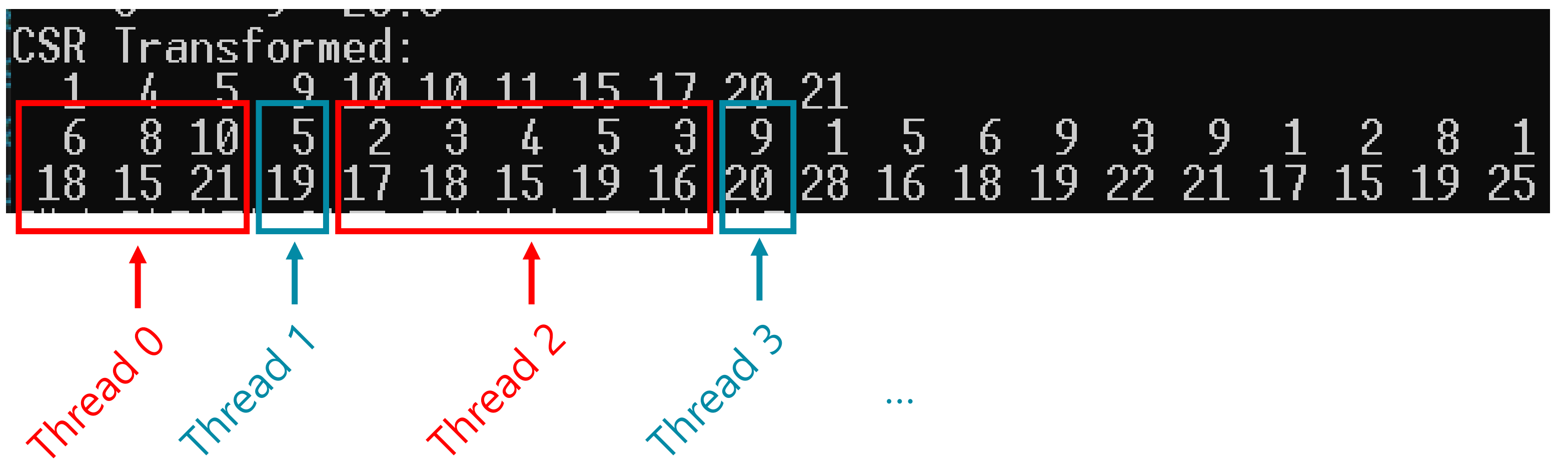
5.4 CSR SpMV using Kernel

```
__global__ void ker_csr_spmv_scalar(
    const int* argJR, const int* argJC, const float* argAA,
    const float* arg_x, float* arg_y) {

    int idx = blockDim.x * blockIdx.x + threadIdx.x;
    float sum = 0;

    for (int i = argJR[idx] - 1; i < argJR[idx + 1] - 1; i++)
        sum += (argAA[i] * arg_x[argJC[i] - 1]);

    arg_y[idx] += sum;
};
```

CUDA Implementation

5.4 CSR SpMV using Kernel

- Straightforward method: Thread per row



```
CSR Transformed:
  1   4   5   9  10  10  11  15  17  20  21
  6   8  10   5   2   3   4   5   3   9   1   5   6   9   3   9   1   2   8   1
 18  15  21  19  17  18  15  19  16  20  28  16  18  19  22  21  17  15  19  25
```

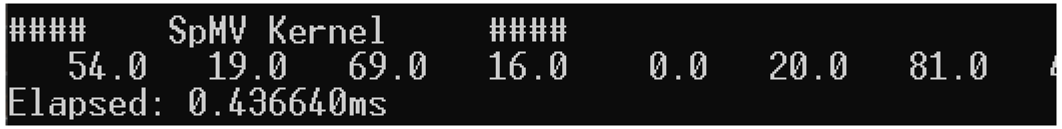
- Poor memory coalescing occurs.
- Unaligned memory access.

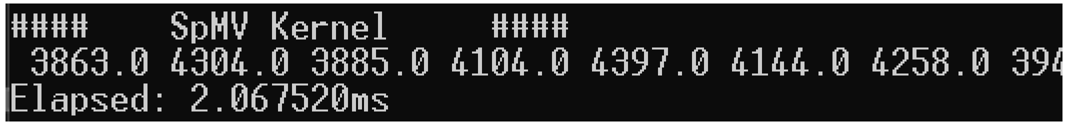
4 CUDA Implementation

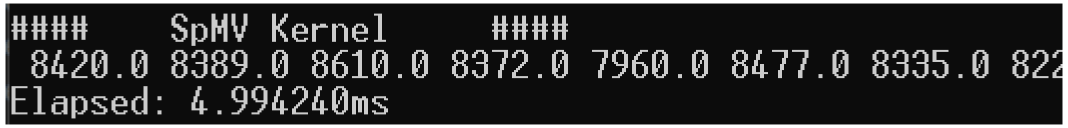
5.4 CSR SpMV using Kernel

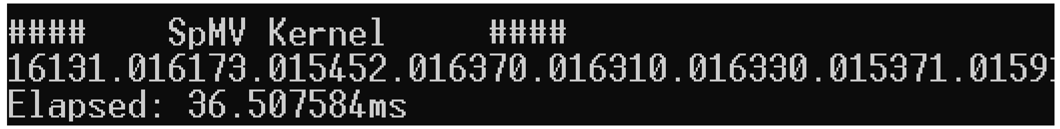
- `ker_csr_spmv_scalar`

Size	NZ	Results

10×10	20	 `####     SpMV Kernel     ####`
		`   54.0    19.0    69.0    16.0     0.0    20.0    81.0`
		`Elapsed: 0.436640ms`

$1{,}024 \times 1{,}024$	209,715	 `####     SpMV Kernel     ####`
		` 3863.0 4304.0 3885.0 4104.0 4397.0 4144.0 4258.0 394`
		`Elapsed: 2.067520ms`

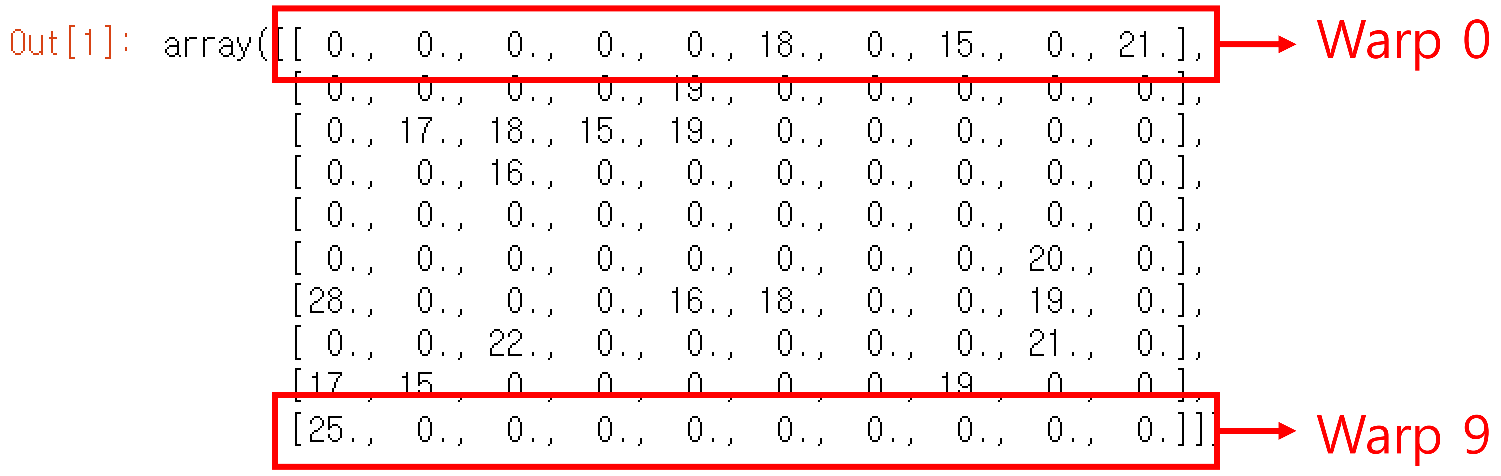
$2{,}048 \times 2{,}048$	838,861	 `####     SpMV Kernel     ####`
		` 8420.0 8389.0 8610.0 8372.0 7960.0 8477.0 8335.0 822`
		`Elapsed: 4.994240ms`

$4{,}096 \times 4{,}096$	3,355,443	 `####     SpMV Kernel     ####`
		`16131.016173.015452.016370.016310.016330.015371.0159`
		`Elapsed: 36.507584ms`

CUDA Implementation

5.4 CSR SpMV using Kernel

- Another method: One warp per row.



```
Out[1]: array([[ 0.,  0.,  0.,  0.,  0., 18.,  0., 15.,  0., 21.],
               [ 0.,  0.,  0.,  0., 19.,  0.,  0.,  0.,  0.,  0.],
               [ 0., 17., 18., 15., 19.,  0.,  0.,  0.,  0.,  0.],
               [ 0.,  0., 16.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
               [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
               [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., 20.,  0.],
               [28.,  0.,  0.,  0., 16., 18.,  0.,  0., 19.,  0.],
               [ 0.,  0., 22.,  0.,  0.,  0.,  0.,  0., 21.,  0.],
               [17., 15.,  0.,  0.,  0.,  0.,  0., 19.,  0.,  0.],
               [25.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.]])
```

- Total threads : $32 \times row$
- Stride : 32

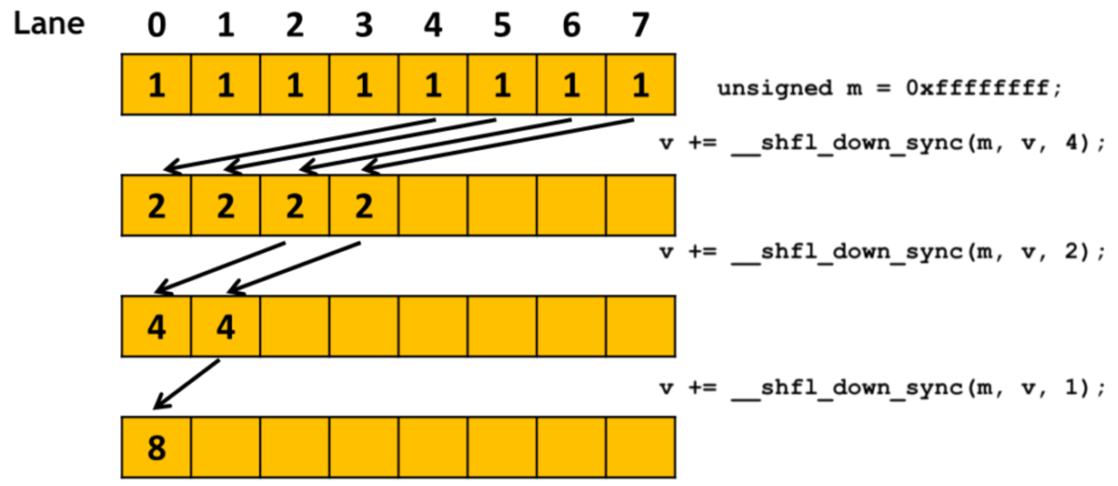
CUDA Implementation

5.4 CSR SpMV using Kernel

```
__global__ void ker_csr_spmv_vector(
    const int* argJR, const int* argJC, const float* argAA,
    const float* arg_x, float* arg_y) {

    int tid= blockDim.x * blockIdx.x + threadIdx.x;
    int wid= tid / 32;
    int lidx= tid & 31;
    float sum= 0;

    for (int i = argJR[wid] - 1 + lidx; i < argJR[wid + 1] - 1; i += 32)
        sum += argAA[i] * arg_x[argJC[i] - 1];

    for (int i = 16; i > 0; i /= 2)
        sum += __shfl_down_sync(0xFFFFFFFF, sum, i);

    if (lidx == 0) arg_y[wid] = sum;
};
```

4 CUDA Implementation

5.4 CSR SpMV using Kernel



```
unsigned m = 0xffffffff;
v += __shfl_down_sync(m, v, 4);
v += __shfl_down_sync(m, v, 2);
v += __shfl_down_sync(m, v, 1);
```

`__shfl_down_sync(FULL_MASK, val, offset)`

- The data exchange is performed between registers, and more efficient than going through shared memory.
- For a thread at lane `x` in the warp, the function gets the value of the `val` variable from the thread at lane `X+offset` of the same warp.
- Available over SM 3.0

https://developer.nvidia.com/blog/using-cuda-warp-level-primitives/

CUDA Implementation

5.4 CSR SpMV using Kernel

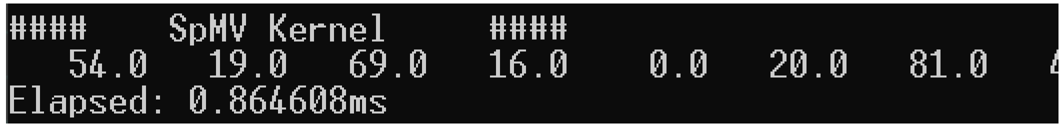
```
__device__  void segmented_reduction(
       const  int lane, const  int * rows, float * vals) {

       //  segmented  reduction  in  shared  memory
    if( lane  >=   1 && rows[threadIdx.x] == rows[threadIdx.x  -   1] )
       vals[threadIdx.x] += vals[threadIdx.x  -   1];
    if( lane  >=   2 && rows[threadIdx.x] == rows[threadIdx.x  -   2] )
       vals[threadIdx.x] += vals[threadIdx.x  -   2];
    if( lane  >=   4 && rows[threadIdx.x] == rows[threadIdx.x  -   4] )
       vals[threadIdx.x] += vals[threadIdx.x  -   4];
    if( lane  >=   8 && rows[threadIdx.x] == rows[threadIdx.x  -   8] )
       vals[threadIdx.x] += vals[threadIdx.x  -   8];
    if( lane  >= 16 && rows[threadIdx.x] == rows[threadIdx.x  - 16] )
       vals[threadIdx.x] += vals[threadIdx.x  - 16];
}
```

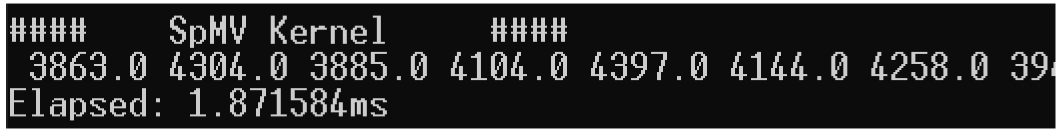
4 CUDA Implementation

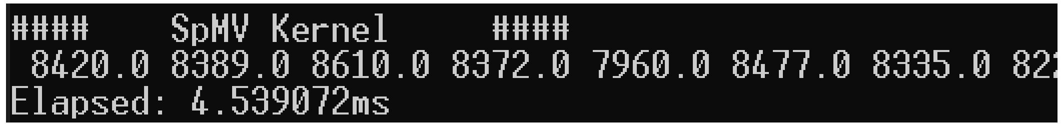
5.4 CSR SpMV using Kernel

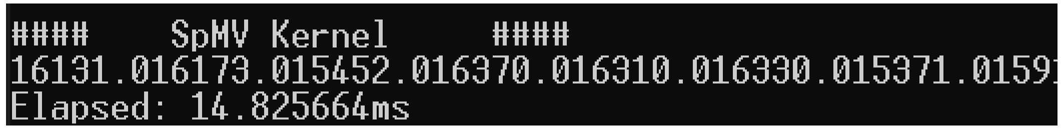
- `ker_csr_spmv_vector`

Size	NZ	Results

10×10	20	 `#### SpMV Kernel ####`
		`54.0 19.0 69.0 16.0 0.0 20.0 81.0`
		`Elapsed: 0.864608ms`

$1{,}024 \times 1{,}024$	209,715	 `#### SpMV Kernel ####`
		`3863.0 4304.0 3885.0 4104.0 4397.0 4144.0 4258.0 39`
		`Elapsed: 1.871584ms`

$2{,}048 \times 2{,}048$	838,861	 `#### SpMV Kernel ####`
		`8420.0 8389.0 8610.0 8372.0 7960.0 8477.0 8335.0 82`
		`Elapsed: 4.539072ms`

$4{,}096 \times 4{,}096$	3,355,443	 `#### SpMV Kernel ####`
		`16131.016173.015452.016370.016310.016330.015371.0159`
		`Elapsed: 14.825664ms`