Batched Sparse Matrix Multiplication for Accelerating Graph Convolutional Networks

Yusuke Nagasaka[†], Akira Nukada[†], Kojima Ryosuke[‡], Satoshi Matsuoka[§,†]

[†]Tokyo Institute of Technology

[‡]Kyoto University

[§]RIKEN Center for Computational Science

Graph Convolutional Networks (GCNs)

- "Graph" is input data of neural network
  - Chemical compounds and protein are expressed as "graph"
  - Knowledge graph

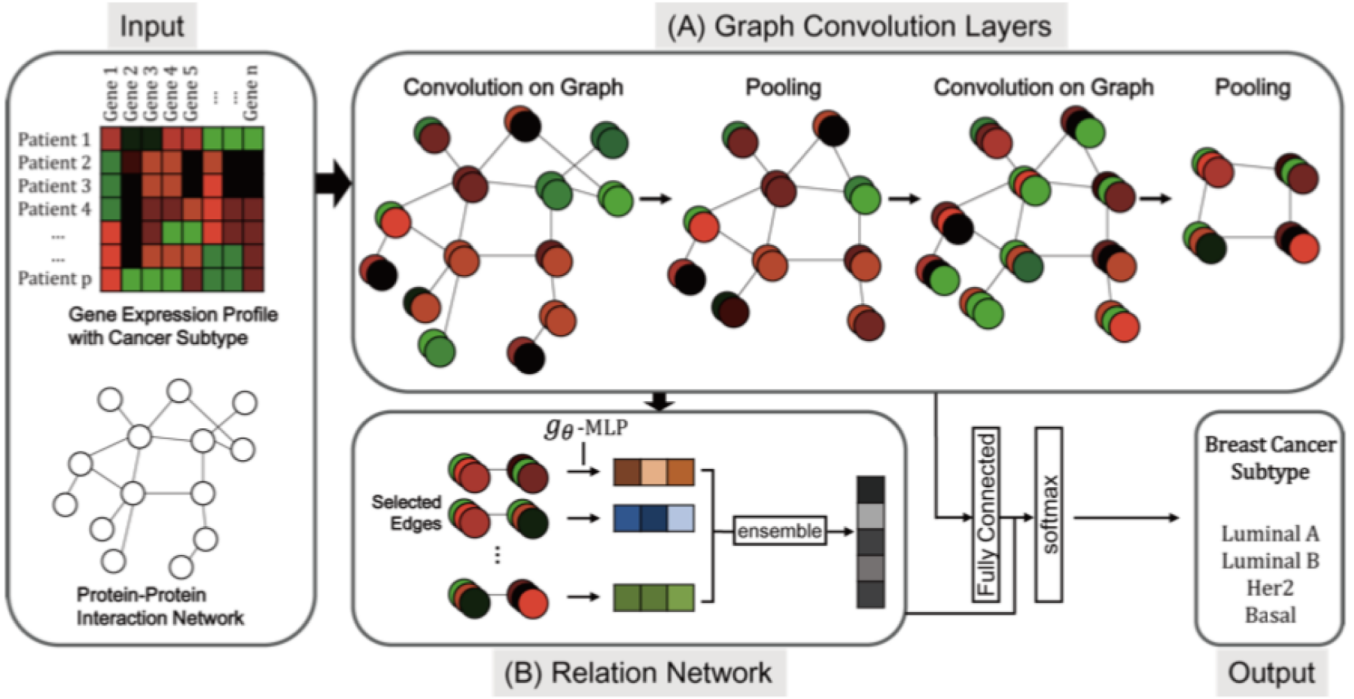


Figure 1: Overview of the proposed method

Quoted from https://arxiv.org/pdf/1711.05859.pdf

Formulation of Graph Convolution

GraphConvolution (Y, A, X, W, bias)

for b ← 0 **to** batchsize
  do for ch ← 0 **to** channel
    do U ← **MatMul** (X[b], W[ch])
      B ← Add(bias[ch], U)
      C[ch] ← **SpMM** (A[b][ch], B)
  Y[b] ← ElementWiseAdd(C)

Execute many SpMM kernels.
Each operation is independent of each other.

Feature 

$$y_i = \sum_j a_{i,j}\, x_j\, W$$

Y = AXW

MatMul and SpMM

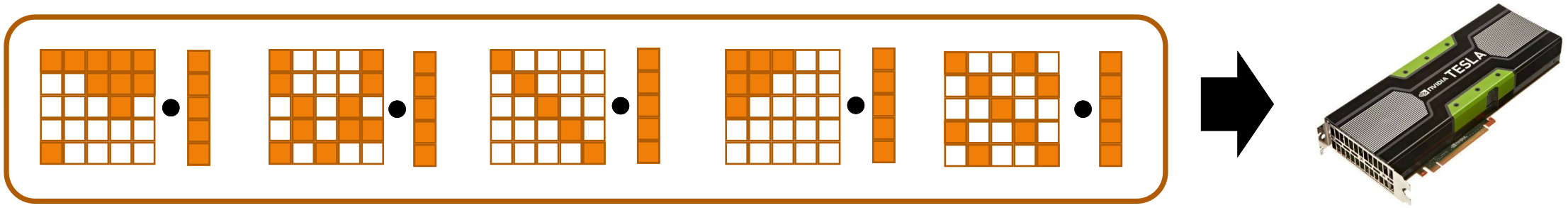
Adjacency matrix

A=

1	0	0	0	0
0	1	0	0	0
1	1	1	1	0
0	0	1	1	1
0	0	0	0	1

Performance Issues of GCNs Applications

- Many small computing kernels occupy the execution time
  - GEMM, Sparse-Dense Matrix Multiplication (SpMM)
    - **Launch overhead of repeated CUDA kernels** is not negligible
  - Not clear how to develop batched (small) sparse matrix routine
    - **Load balance issue**
      - Number of nodes / sparsity of graph varies by input graphs
    - Occupancy issue
      - Require architecture specific kernel
- ***How to efficiently compute tens or hundreds of small SpMMs?***

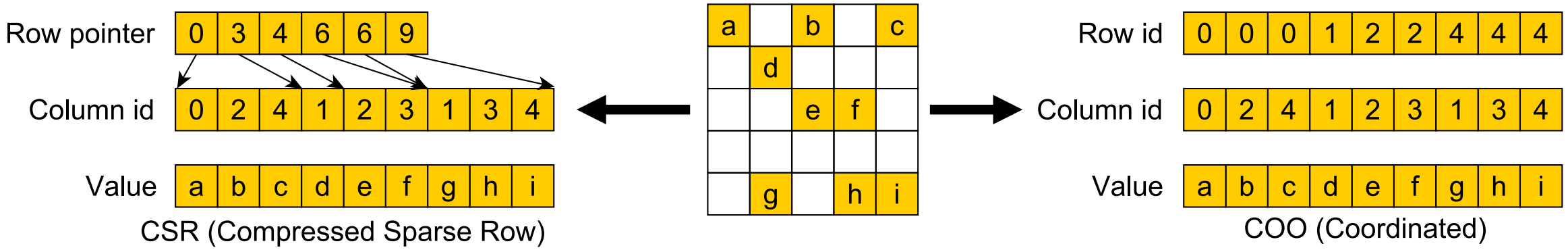


Contribution

- Batched approaches for SpMM on GPU to improve the performance of GCNs applications
  - **Sub-Warp-Assigned (SWA) SpMM** for SpMM on small matrices
    - Support both SparseTensor and CSR
  - **Batched SpMM**
    - High occupancy and utilization of fast shared memory
    - Reduce the overhead of CUDA kernel launches
    - Develop routines both for SparseTensor and CSR
  - Execute tens or hundreds small SpMM by single kernel
  - **Significant performance boost**
    - Up to **9.27x** speedup compared to Non-batched approaches for SpMM
    - **Up to 1.59x speedup** for training and **1.37x speedup** for inference on GCNs application

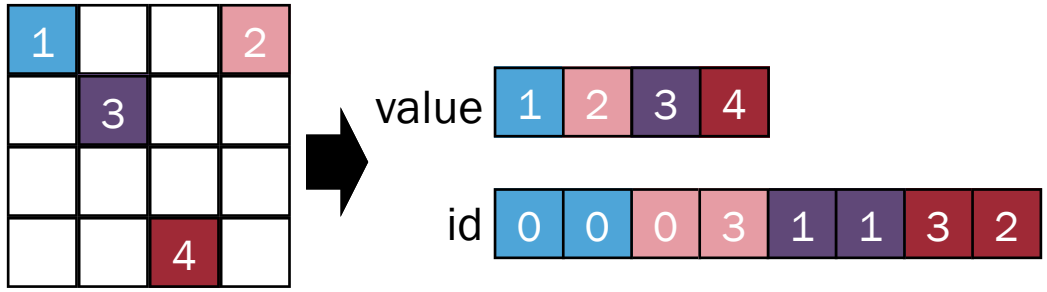
Sparse Matrix Format

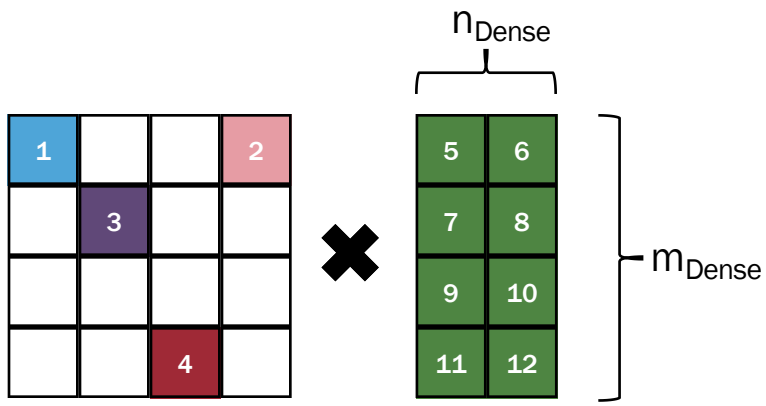
- Compressing needless zero elements
  - Storing only non-zero elements
  - Reducing memory usage and computation
- Many formats have been proposed
  - Being suited to architectures and given matrices

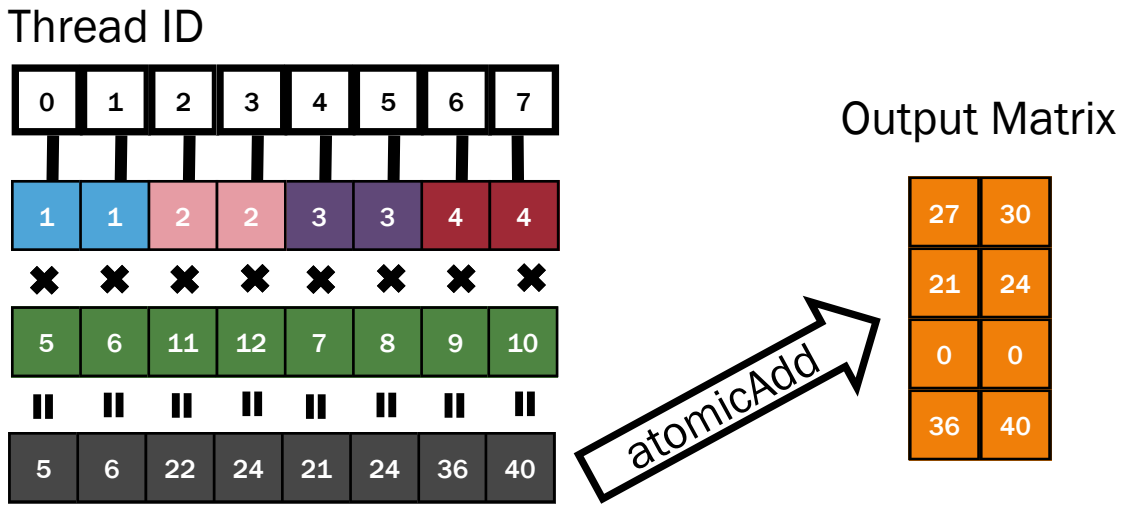


Implementation of SpMM in TensorFlow

- COO-like sparse matrix format
  - Array of {Row, Column} ids
- SparseTensorDenseMatmul
  - 1 CUDA thread for 1 mul-add operation
  - nnz * n_{Dense} threads
  - Load-balanced workload
  - Addition is done by atomicAdd
    - **Expensive on global memory**







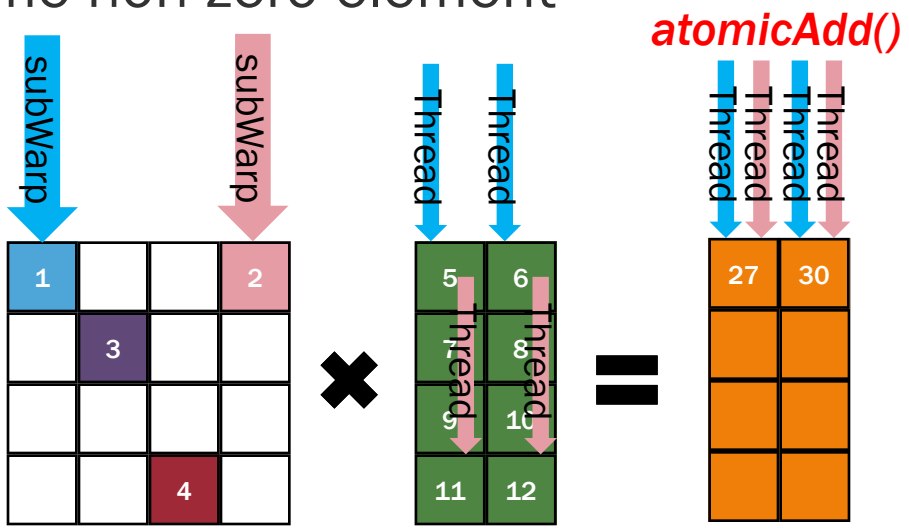
Sub-Warp-Assigned (SWA) SpMM

- Assign *subWarp* to each non-zero element
  - *subWarp* is set as power of two
    - **Division and modulo operations by executing low-cost bit operations**

$$subWarp = \begin{cases} 32 & (n_B > 16) \\ \min 2^p \ s.t.\ n_B \le 2^p & (n_B \le 16) \end{cases}$$

  - Reduce instructions for memory access to same non-zero element

SWA_SpMM (C, A, B, subWarp)
// set matrix C to 0
i ← threadId
nzid ← i /subWarp
rid ← ids_A[nzid ∗ 2]
cid ← ids_A[nzid∗2+1]
val ← $values_A$[nzid]
for j ← (i % subWarp) **to** n_B **by** subWarp
  do Atomic (C[rid][j] ← C[rid][j] + val ∗ B[cid][j])

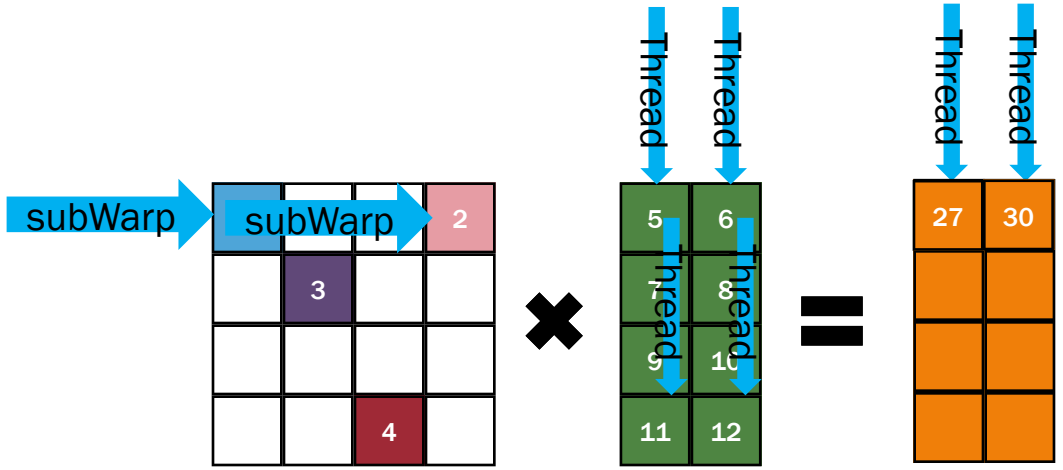


Sub-Warp-Assigned (SWA) SpMM for CSR

- Assign *subWarp* to each row of input sparse matrix
  - Reduce instructions for memory access to same non-zero element
  - **Atomic-free addition to output matrix**

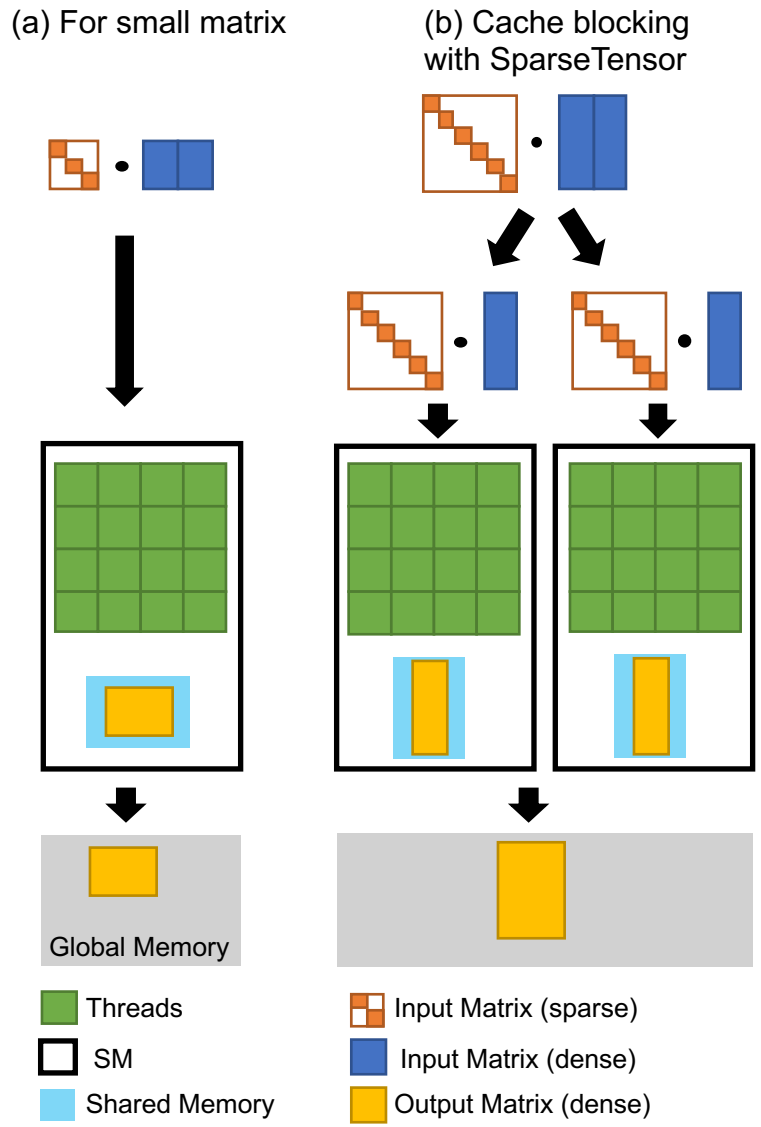
SWA_SpMM_CSR (C, A, B, subWarp)

```
// set matrix C to 0
i ← threadId
rid ← i / subWarp
for nzid ← rpt_A[rid] to rpt_A[rid + 1]
    do cid ← colids_A[nzid]
        val ← values_A[nzid]
        for j ← (i % subWarp) to n_B by subWarp
            do C[rid][j] ← C[rid][j] + val * B[cid][j]
```



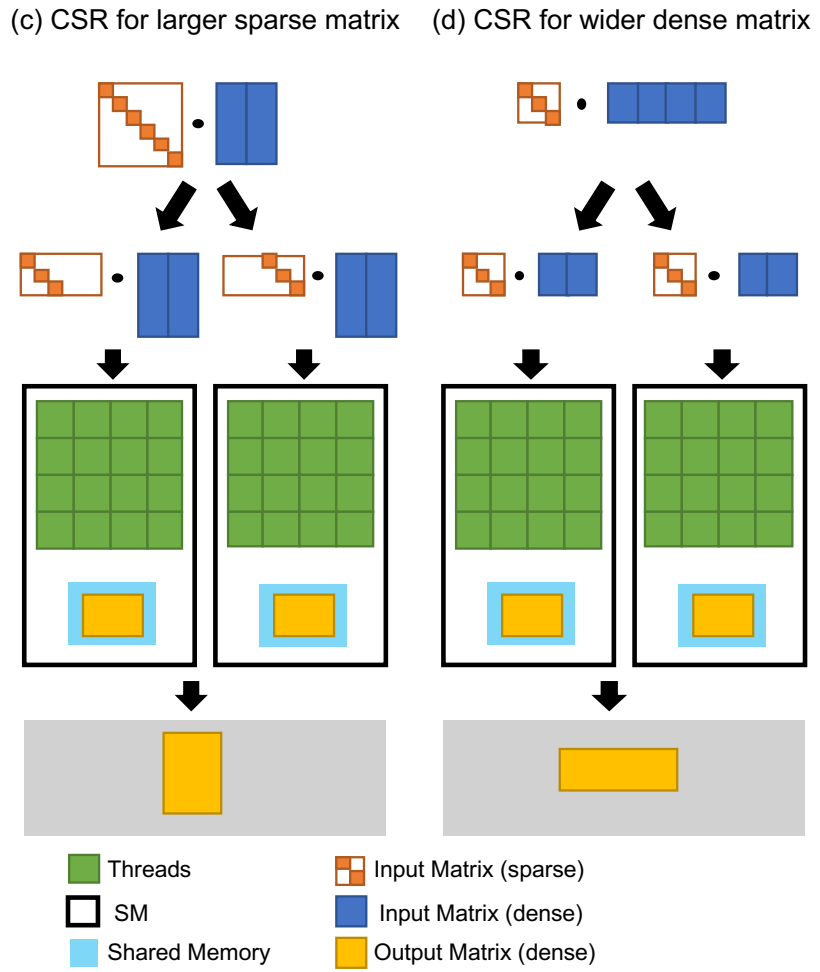
Efficient Use of Shared Memory

- Utilize shared memory for output matrix
  - **Reduce the overhead of CUDA kernel launch** for initializing output matrix
  - **Hardware support for atomic operation** on shared memory
- Cache blocking optimization for larger inputs
  - Divide the output matrix along the column
    - **Larger output matrix can be placed on shared memory**
    - Also **improve the locality of memory access to input dense matrix**



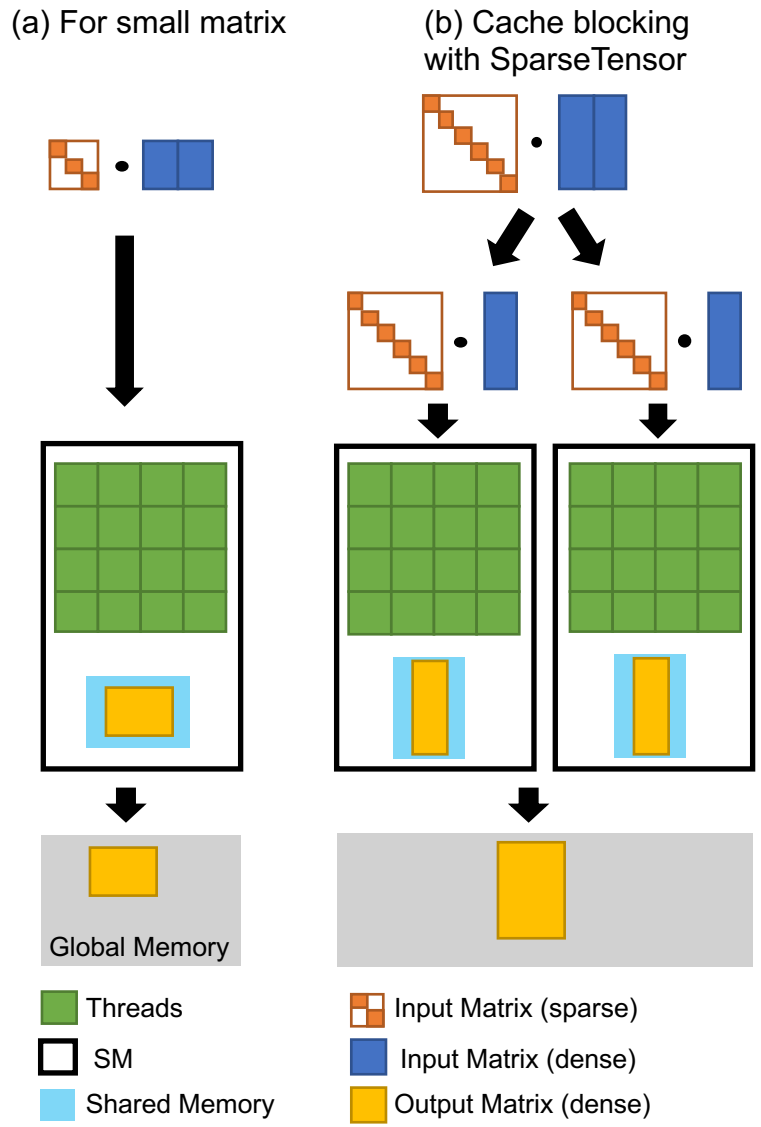
Efficient Use of Shared Memory for CSR

- Each subWarp keeps its output row (= n_B)
  - Not need to keep whole output matrix (= m_A * n_B) by each thread block
- More thread blocks for larger m_A
  - subWarp * m_A > TB
  - Row-wise division of input sparse matrix
- Cache blocking for wider dense matrix
  - TB / subwarp * n_B > 32KB
    - Capacity of shared memory is 32KB
    - TB is thread block size
  - **Improve the locality of memory access to input dense matrix**



Batched Algorithm for SpMMs

- 1 CUDA kernel manages multiple SpMMs
  - **Reduce the overhead of CUDA kernel launch**
- Statically decide whether cache blocking optimization is applied
  - Select (a) or (b) based on maximum size of output
- Assign one thread block to each SpMM for whole matrix or sub matrix



Performance Evaluation

Evaluation Environment

- TSUBAME 3.0
  - CPU: Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz
  - GPU: NVIDIA Tesla P100
    - #SM: 56
      - Shared memory: 64 KB/SM
    - Memory: 16GB
  - SUSE Linux Enterprise
  - NVCC V9.0.176

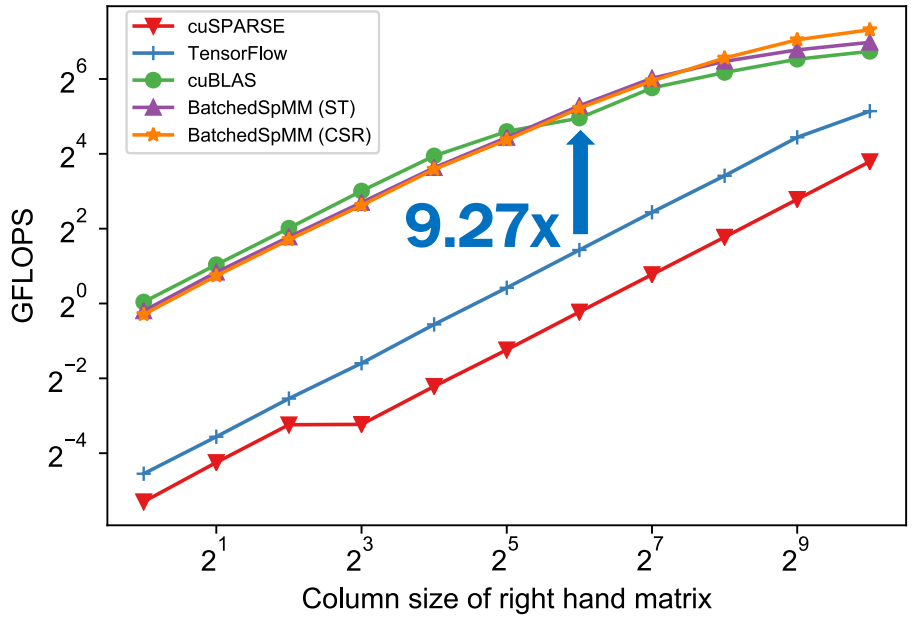
Benchmark of Batched Approaches for SpMM

- Compare the performance of
  - *csrmm*() and *csrmm2*() in cuSPARSE (non-batched)
  - SpMM following SparseTensorDenseMatMul in TensorFlow (non-batched)
  - *gemmBatched*() from Batched BLAS (batched, but for dense x dense MM)
  - Batched SpMM for SparseTensor (batched)
  - Batched SpMM for CSR (batched)
- Randomly generate sparse matrix
  - Parameter: Row/column size (= dim), sparsity (= nnz/row), batch
- FLOPS in single precision
  - $2 * nnz_A * n_B$ / exe_time
  - Not include the operations between zero elements in *gemmBatched*()

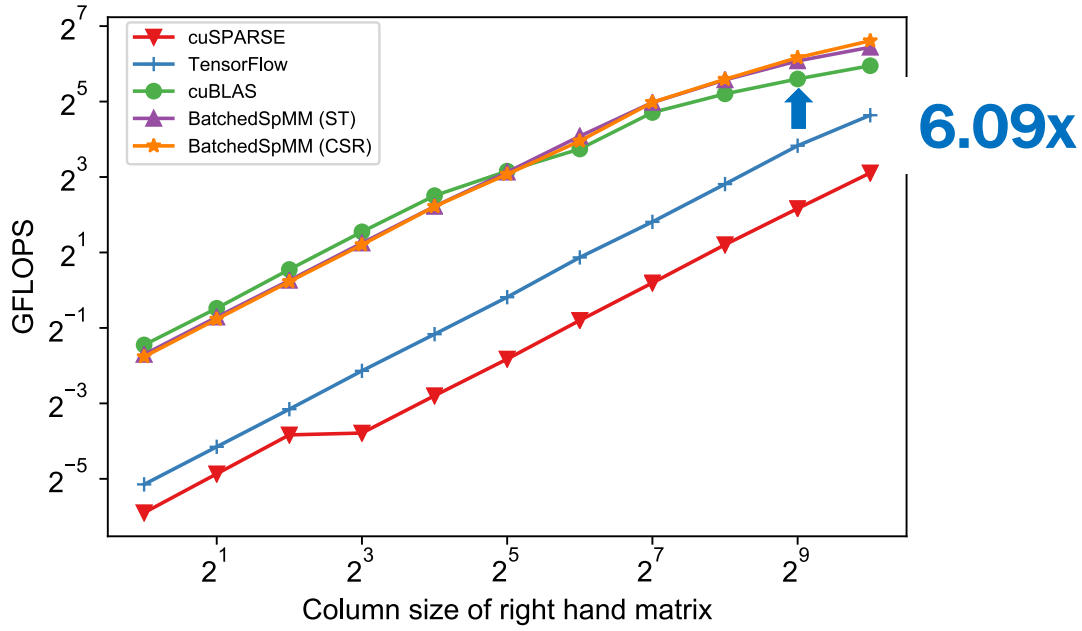
Benchmark Results

- Parameter settings are based on dataset and configuration of GCNs application
- **Significant speedups by Batched SpMM family**
  - Better *sm_efficiency* with Batched SpMM

TensorFlow	BatchedSpMM (ST)	BatchedSpMM (CSR)
35.51%	**89.07%**	**87.87%**



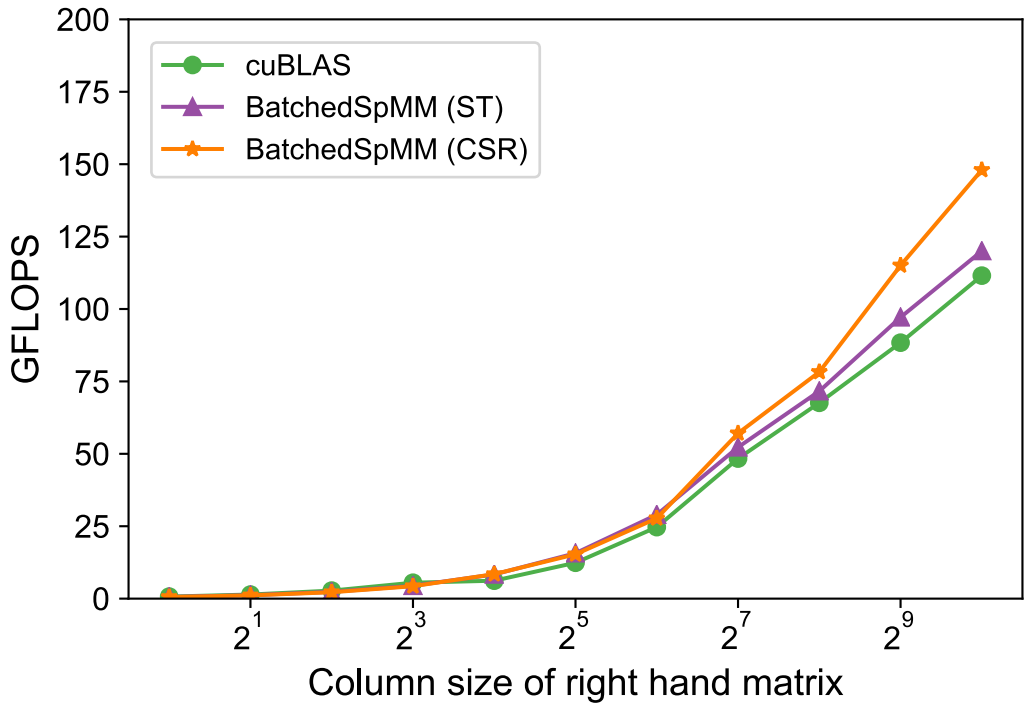
batch=50, dim=50, nnz/row=2



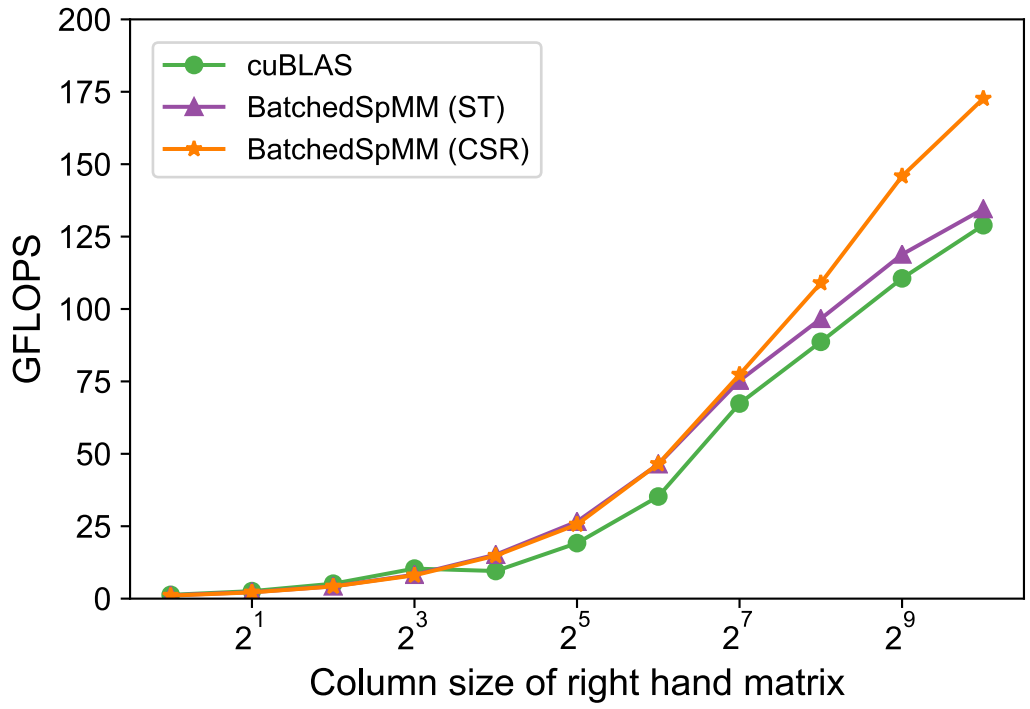
batch=100, dim=50, nnz/row=3

Benchmark Results
Batch size

- Precise comparison between batched approaches
- Larger batch size simply brings higher throughput of SpMMs
  - Batch=50 cases do not use all SMs on GPU



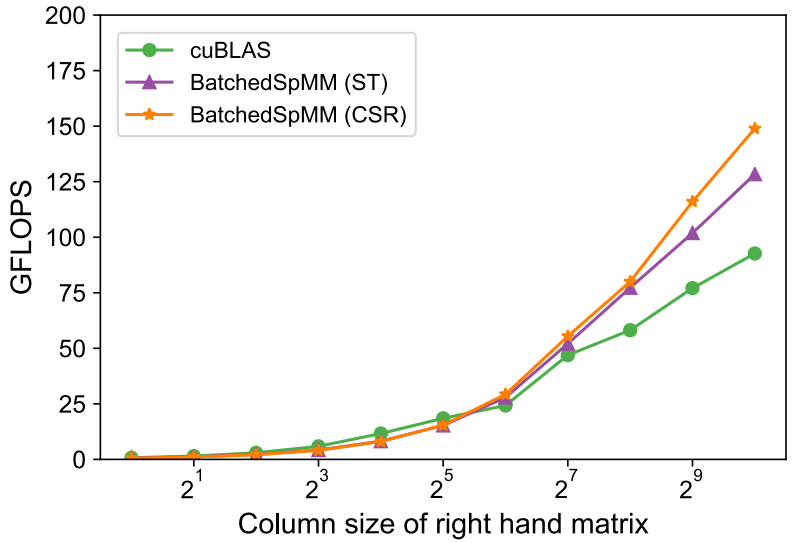
batch=50, dim=64, nnz/row=3



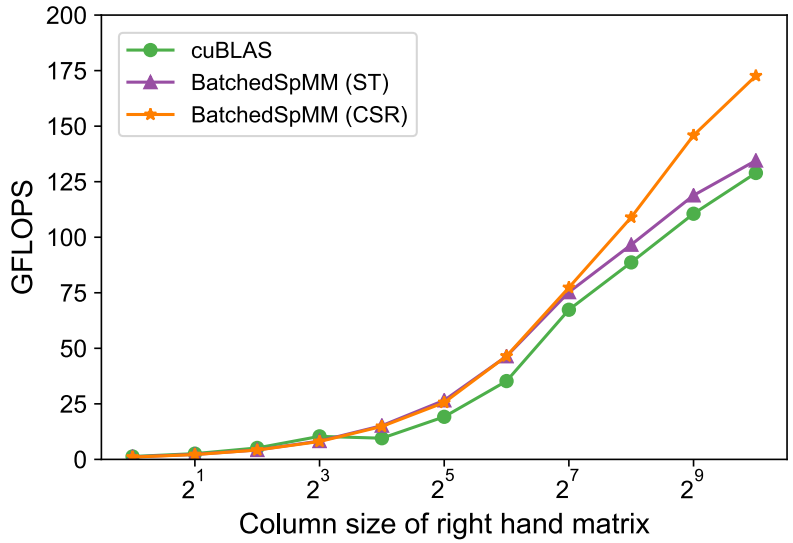
batch=100, dim=64, nnz/row=3

Benchmark Results
Dimension

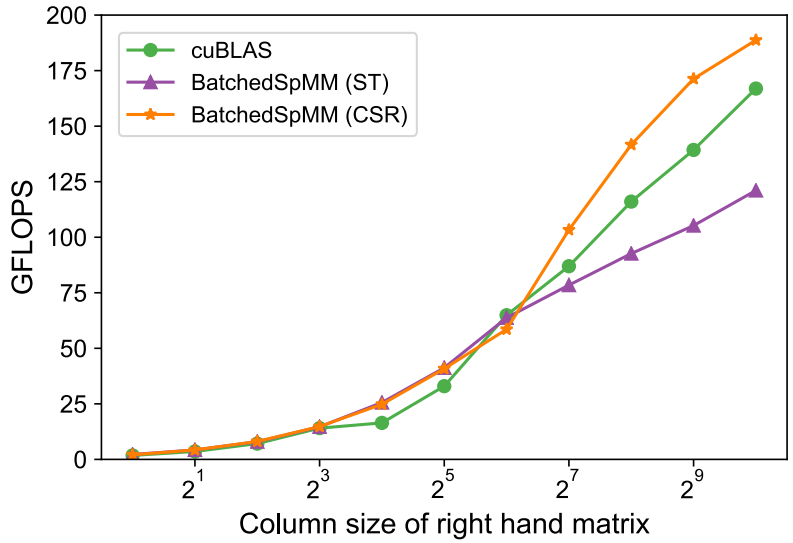
- BatchedSpMM (CSR) is getting better performance
  - Improvement of parallelism
    - Batched SpMM for CSR launches more threads in proportion to m_A
- **Improvement of cuBLAS and BatchedSpMM (ST) is limited**
  - Increase of dim results in increase of sparsity, more zero-related operations
  - More cache blocking causes memory pressure to same non-zero element



batch=100, dim=32, nnz/row=3



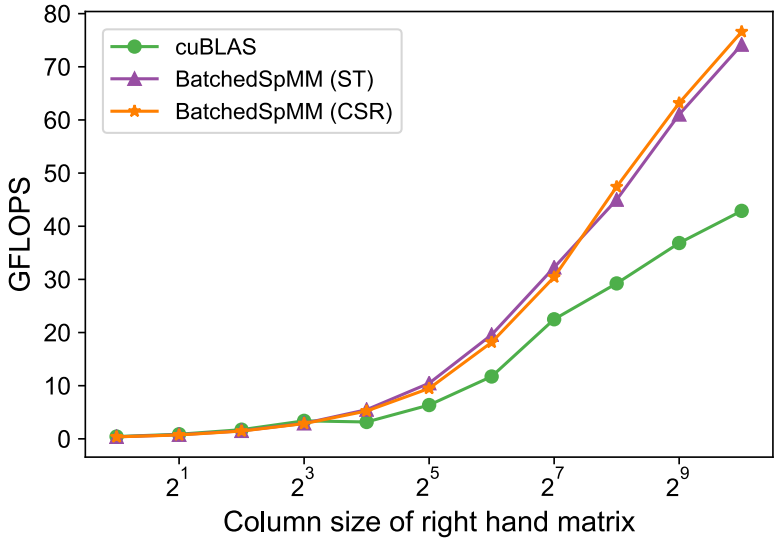
batch=100, dim=64, nnz/row=3



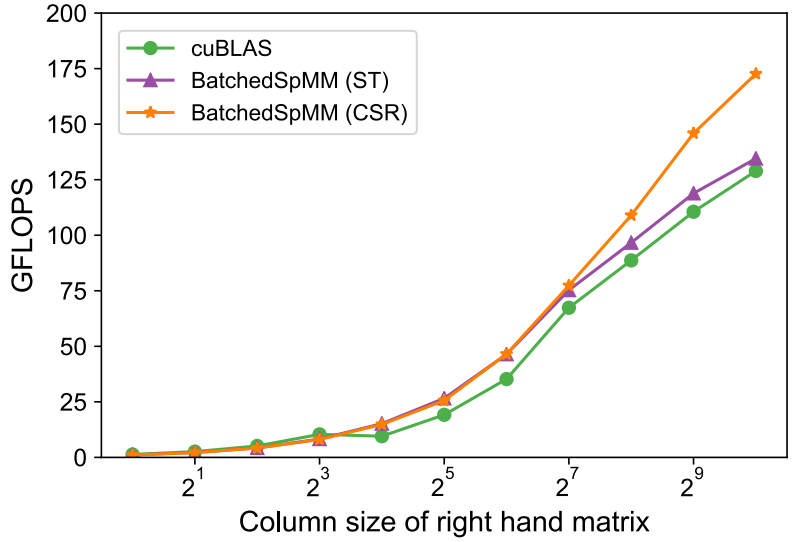
batch=100, dim=128, nnz/row=3

Benchmark Results
Sparsity

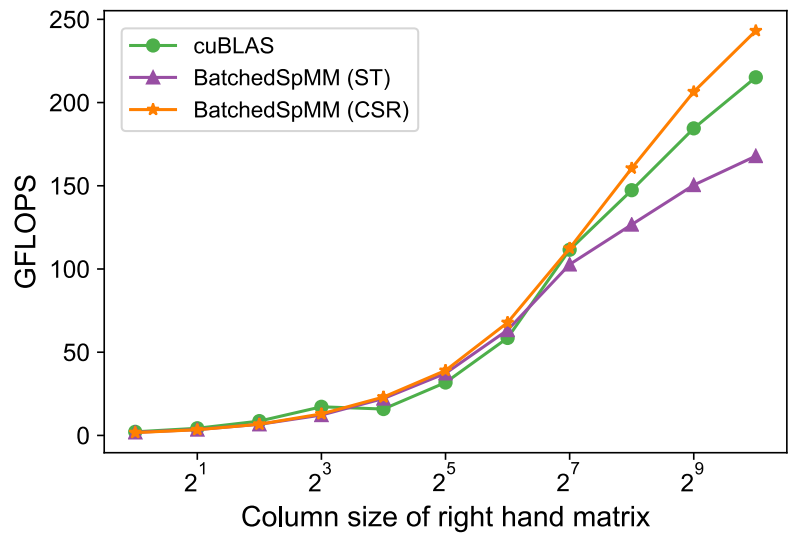
- Batched SpMM kernels work efficiently on sparser matrices
  - Improvement of Batched SpMM (ST) is limited
    - More race condition by atomic operation
- cuBLAS appears to show better performance on denser matrices



batch=100, dim=64, nnz/row=1



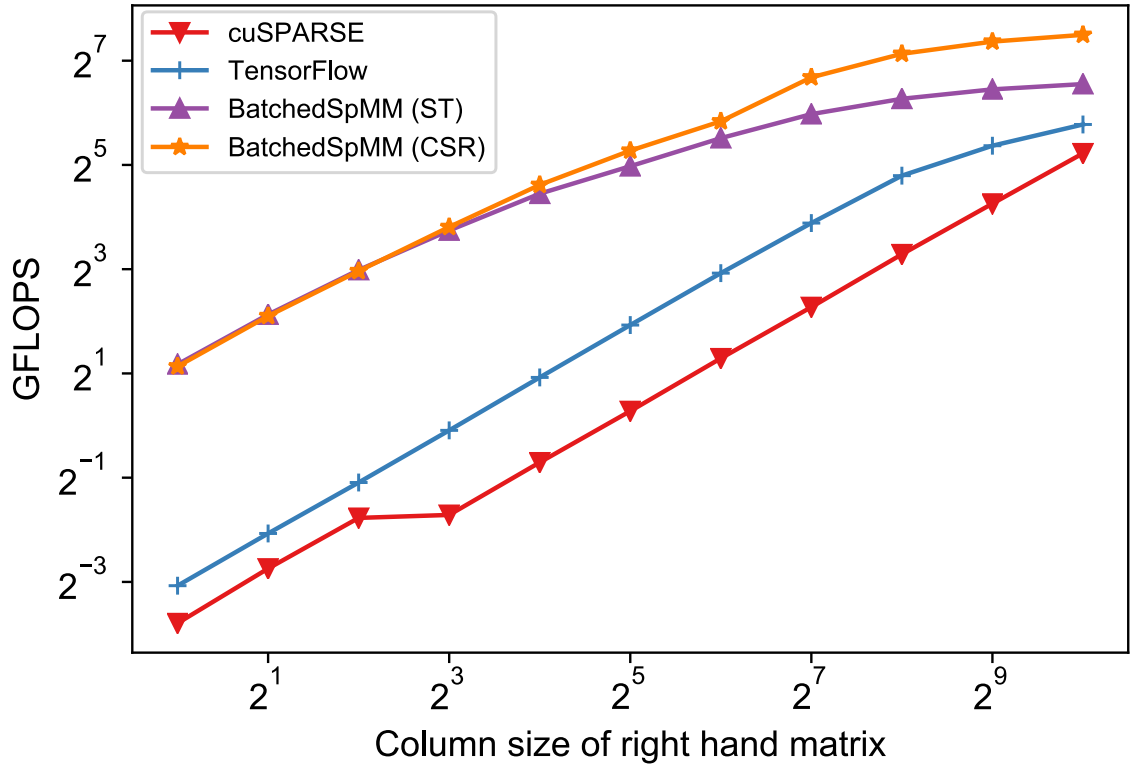
batch=100, dim=64, nnz/row=3



batch=100, dim=64, nnz/row=5

Benchmark Results
Mixed

- Various inputs with changing dimension and sparsity
  - dim = [32, 256], nz/row = [1, 5], batch = 100
  - cuBLAS is excluded because it requires same input matrices sizes
  - **3.29x** performance improvement at n_B=1024



Evaluation on GCNs Application

- ChemGCN implemented with TensorFlow
- Dataset and configuration

	#Matrices	Max Dimension	Epoch	Batch size (Training / Inference)	#layer of GraphCNN
Tox21	7,862	50	50	50 / 200	2
Reaction100	75,477	50	20	100 / 200	3

- Average time of 5 executions

Formulation of Graph Convolution (again)

GraphConvolution (Y, A, X, W, bias)

for b ← 0 **to** batchsize
  do for ch ← 0 **to** channel
    do U ← MatMul (X[b], W[ch])
      B ← Add(bias[ch], U)
      C[ch] ← SpMM (A[b][ch], B)
  Y[b] ← ElementWiseAdd(C)

GraphConvolutionBatched (Y, A, X, W, bias)

for ch ← 0 **to** channel
  do X_r ← Reshape(X, (m_x * batchsize, n_x)
    U ← MatMul(X_r, W[ch])
    B ← Add(bias[ch], U)
    A_{list} ← [A[0][ch], … , A[batchsize – 1][ch]
    C[ch] ← BatchedSpMM(A_{list}, B)
Y ← ElementWiseAdd(C)

Evaluation on GCNs Application

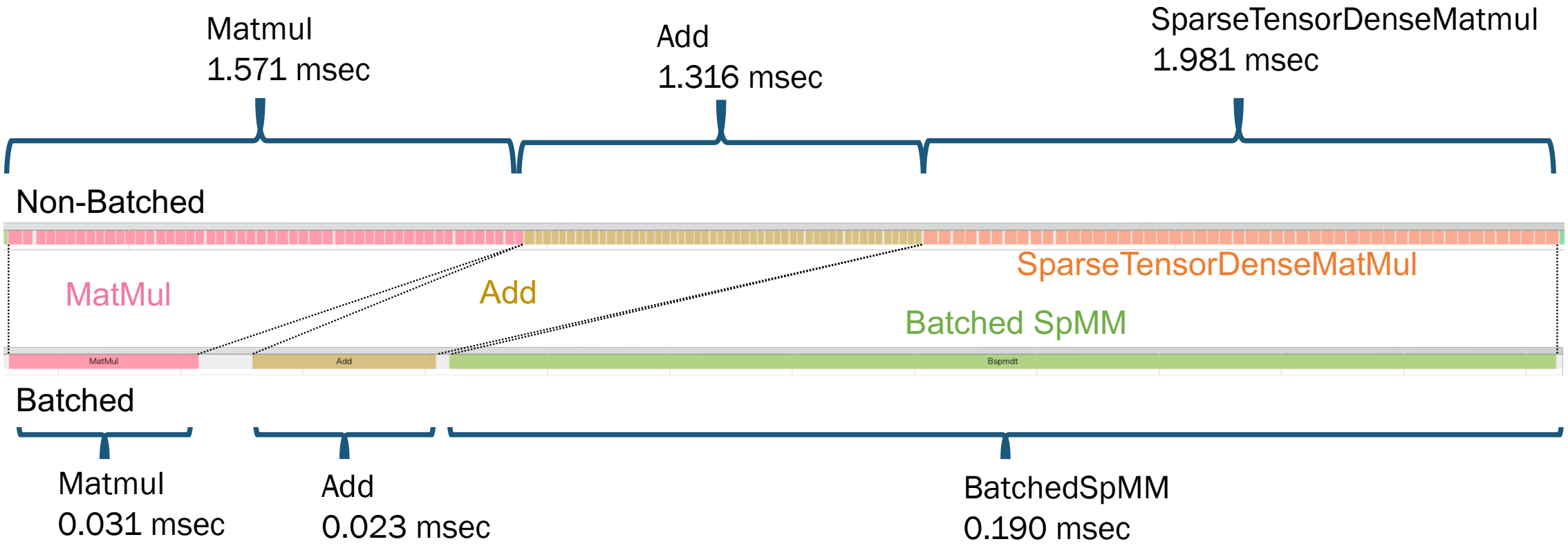
- Batched SpMM is used as Batched version
  - Training: **Up to 59% improvement**
  - Inference: **Up to 37% improvement**
  - Data of Tox21 can be placed on LL cache in CPU case

		CPU	GPU		
		Non-Batched	Non-Batched	Batched	Speedup
Training	Tox21	854.51	918.03	723.80	1.18x
	Reaction100	16223.98	3029.13	1905.32	1.59x
Inference	Tox21	2.71	2.56	1.97	1.30x
	Reaction100	44.66	22.42	16.32	1.37x

Execution Time [sec]

Profiling with Timeline

- Profiling result of GraphConvolution layer with Tox21 data
- **Reduction of kernel launches**
  - CUDA kernel launches: 50 * 3 => 3



Related Work

- Batched BLAS
  - Handles many operations on dense matrix or vector in a single kernel
  - **High throughput for kernels on small matrices**
  - Batched SpMV
    - **Highly application specialized (e.g. assumes same non-zero pattern)**
- Libraries and Framework for GCNs
  - DeepChem
    - Graph structure is expressed as adjacency list
  - Chainer Chemistry
    - Treat sparse matrix as dense matrix
      - **Many zero-related operations**

Conclusion

- Efficient algorithms for many SpMM operations for small matrix
  - **Sub-Warp Assigned SpMM**
  - **Batched SpMM**
    - Improve the locality of memory access and exploit shared memory
- Significant performance boost
  - Detailed preliminary performance evaluation
    - **Up to 9.27x speedup** from Non-batched SpMM kernel
    - Performance advantage to Batched GEMM for small matrices
  - Evaluation on GCNs application
    - **Up to 1.59x speedup** for training and **1.37x speedup** for inference

Code will be ready in the end of May
https://github.com/YusukeNagasaka/Batched-SpMM