This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

SAND2021-2511C

Sandia National Laboratories

Exceptional service in the national interest

Mixed-Precision Schemes for Linear Algebra Kernels on GPUs

SIAM CSE ‘21

Siva Rajamanickam, Gordon Moon*

Scalable Algorithms Department, Center for Computing Research

*Currently at Korea Aerospace University

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy’s National Nuclear Security Administration under contract DE-NA0003525.

Outline

Utilizing low-precision hardware for SpMM

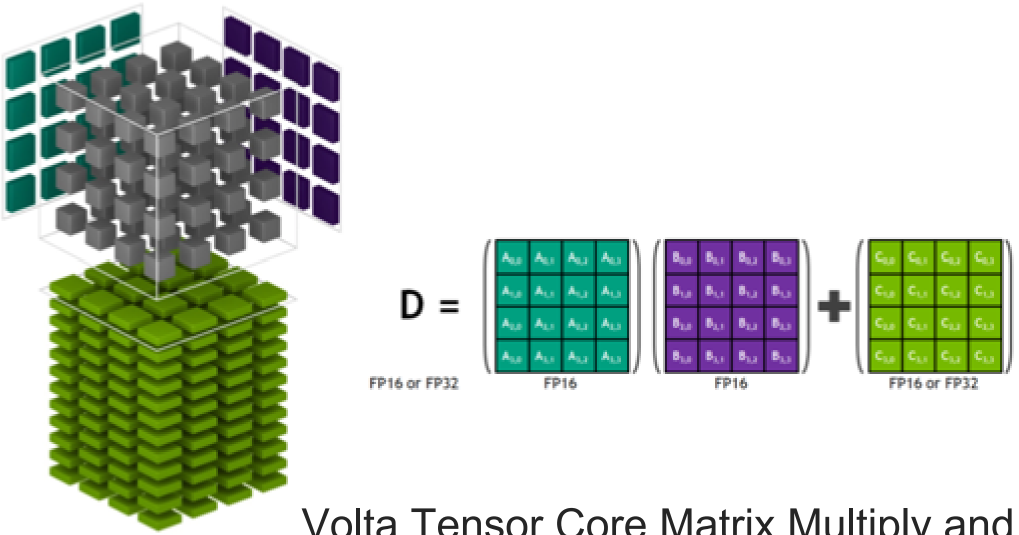
- Use existing hardware (GPUs) to evaluate algorithmic approaches
- Can we use current low-precision hardware for sparse-matrix methods?

Simulations of future hardware for GEMMs

- How do we generalize to different block sizes?
- How can we study different GEMM sizes on different types of accelerators?
- Can we study (using simulations) other hardware for GEMMS?

Motivation

NVIDIA Volta GPU uses tensor cores to achieve good performance on dense matrix-dense matrix product



Volta Tensor Core Matrix Multiply and Accumulate*

Can we use the tensor cores as the example for future spatial accelerators ?

How to utilize tensor cores to efficiently perform sparse-matrix multivector product?

*Source: https://developer.nvidia.com/blog/tensor-core-ai-performance-milestones/

Related Work

Multiprecision / Mixed-precision

- This two part mini-symposium **and** a parallel mini-symposium

Tiled Sparse Matrix times dense vector multiplication

- Decades of work in sparse matrix ordering techniques
- Decades of work in sparse matrix partitioning

Most recent work on Tiling strategies

- Hong et al. **Adaptive sparse tiling for sparse matrix multiplication,** PPoPP, 2019. (ASpT-RR)
- Jiang et al. **A novel data transformation and execution strategy for accelerating sparse matrix multiplication on GPUs,** PPoPP 2020, (ASpT-NR)
- Spatial partitioning approaches for graph algorithms (Yasar, Catalyurek et al.) – For future, not considered here

Several Use Cases

Block matrices are becoming more popular (again) with multiphysics use cases

- Block sizes are dictated by physics, not the clean power of two that the hardware likes
- Simple solution is to do padding and work on zeros.

Use Tiled SpMM for general matrices

- Use ordering methods to find “small” blocks in reordered matrices
- Could lead to variable sized blocks if ordering is not targeted for tiling
- Pad the blocks to arrive at uniform set of blocks that can be used with the hardware

Can we utilize the tiling / ordering approaches to find dense blocks for tensor cores?
Can we utilize the natural blocks that occur in Multiphysics use cases ?

Simple SpMM for general and Multiphysics use cases

General use case

Phase 1: Perform Reverse Cuthill–McKee (RCM) to re-order a symmetric/unsymmetric sparse matrix (after symmetrizing)

Phase 2: Divide the re-ordered sparse matrix into diagonal block sub-matrices and off-diagonal elements

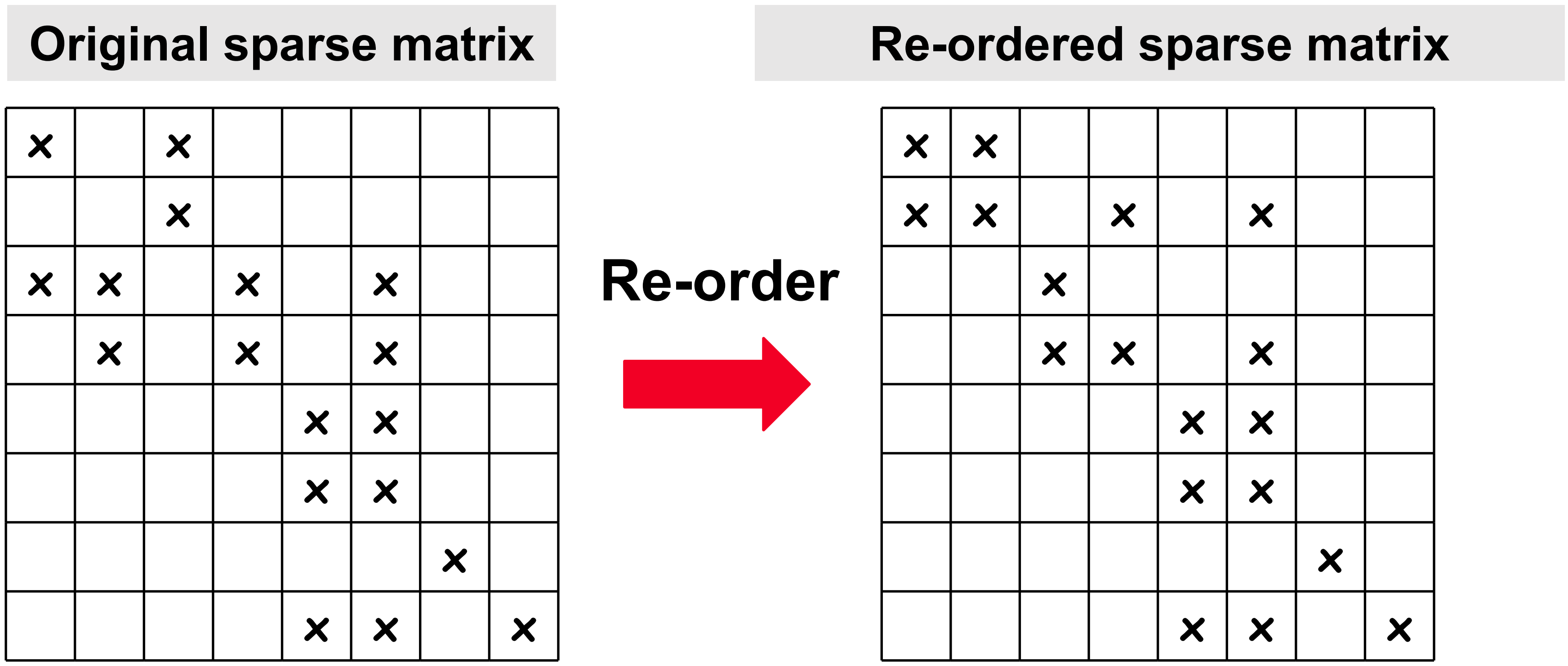
Phase 3

- Run dense diagonal block sub-matrices on tensor cores
- Run off-diagonal sparse matrix on regular SMs
- Accumulate the result matrices computed by tensor cores and regular SMs

Multiphysics use case

- No need for ordering
- No need for splitting the matrix
- Sparse matrix with each scalar replaced by a dense block (need indirect indexing to get the dense blocks)

SpMM Sparse + Dense: Phase 1



Original sparse matrix

×		×					
		×					
×	×		×		×		
	×		×		×		
				×	×		
				×	×		
						×	
				×	×		×

Re-order

Re-ordered sparse matrix

×	×						
×	×		×		×		
		×					
		×	×		×		
				×	×		
				×	×		
						×	
				×	×		×

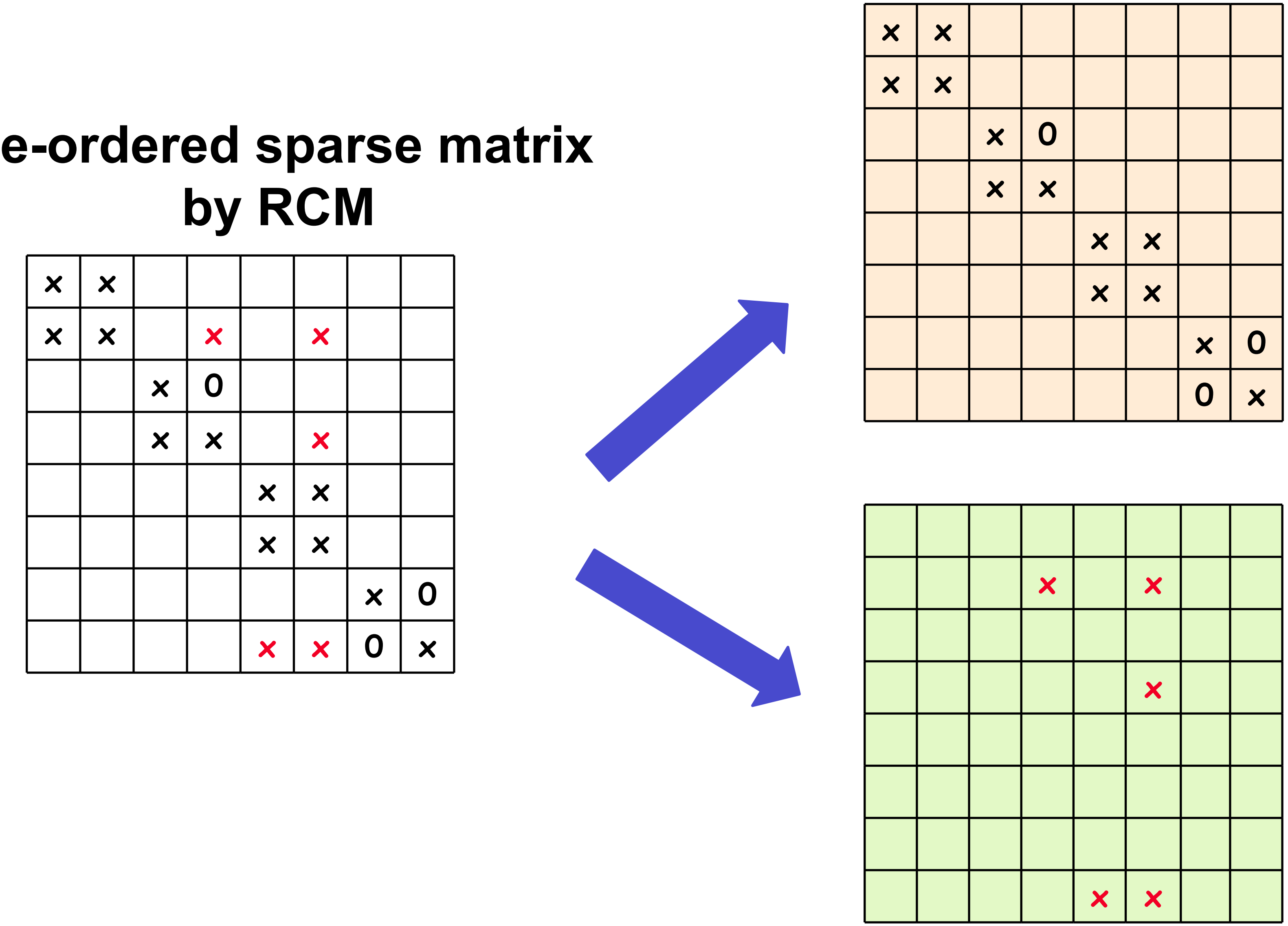
permuted row/column index order

e.g., P = [1,3,2,4,5,6,7,8]

×: non-zero element

SpMM Sparse + Dense: Phase 2

Re-ordered sparse matrix by RCM



×	×						
×	×		×		×		
		×	0				
		×	×		×		
				×	×		
				×	×		
						×	0
				×	×	0	×

×	×						
×	×						
		×	0				
		×	×				
				×	×		
				×	×		
						×	0
						0	×

Each diagonal block size = 2 × 2

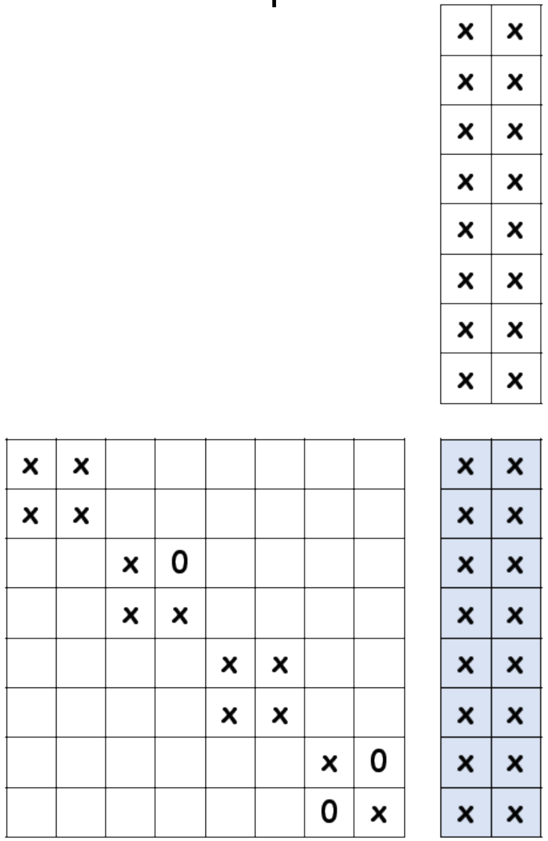
Running on tensor cores

			×		×		
					×		
				×	×		

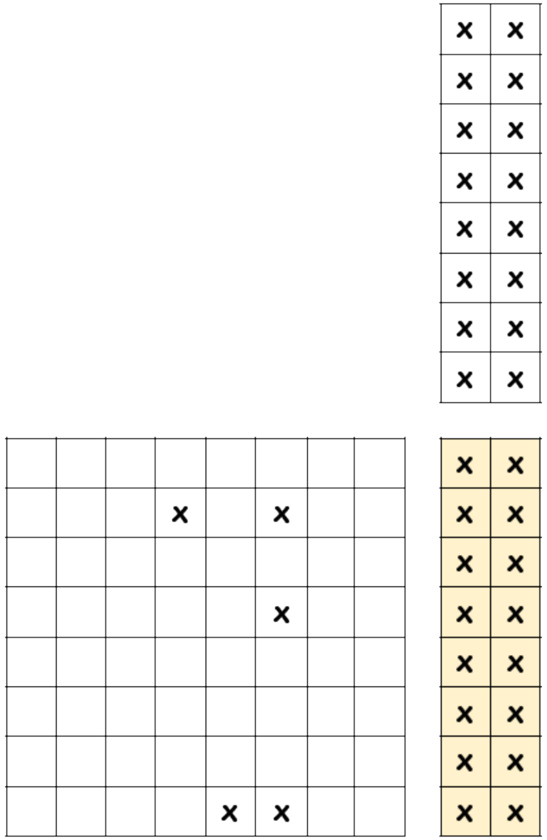
Running on regular SMs (cuSPARSE)

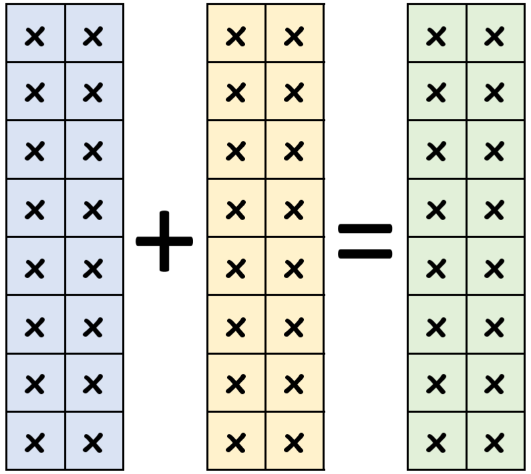
SpMM Sparse + Dense: Phase 3

Running on tensor cores – multiple block-wise GEMM operations



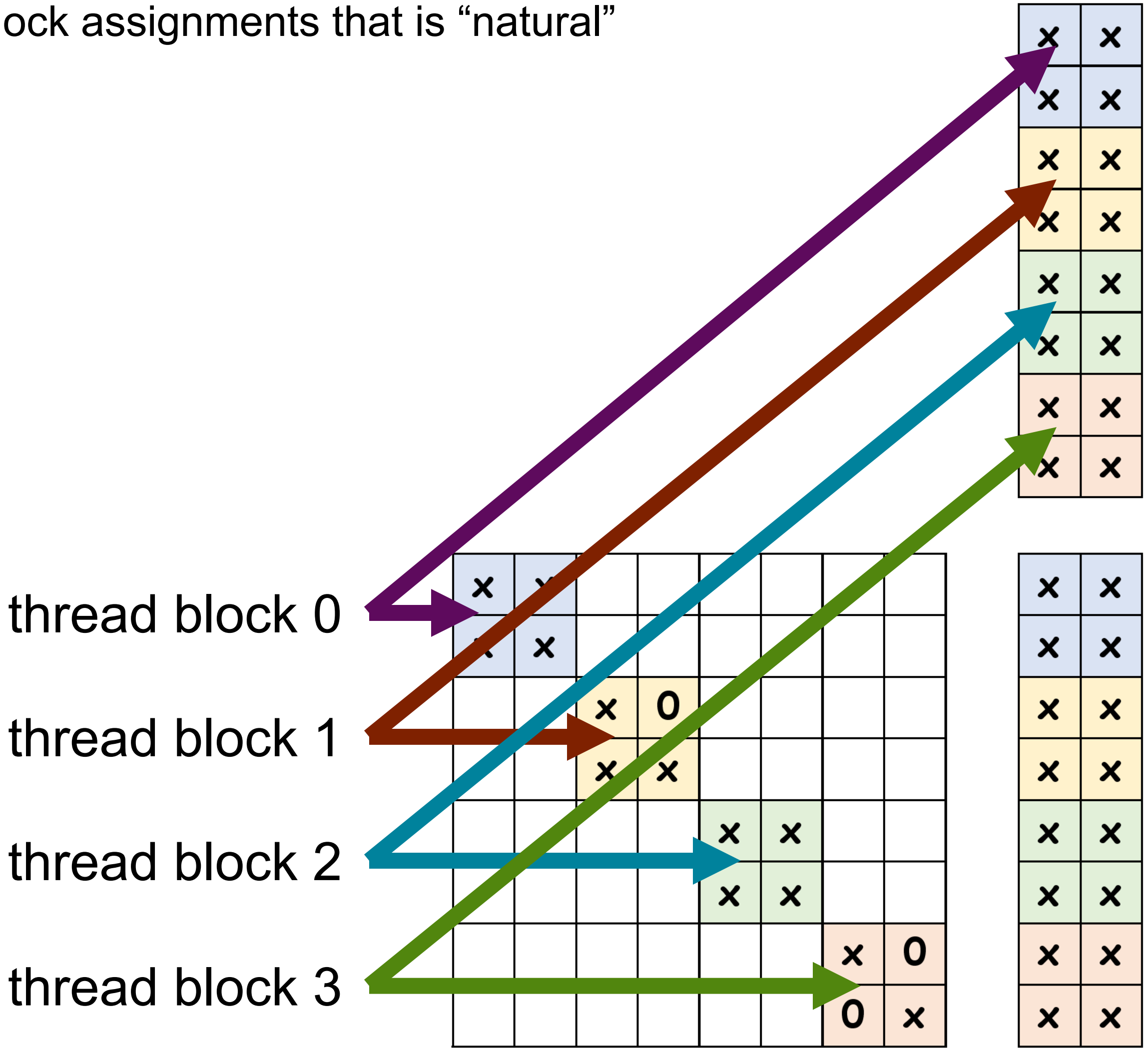
Running on regular SMs – SpMM





Phase 3 – Block-wise GEMM operations

Do a thread-block assignments that is “natural”



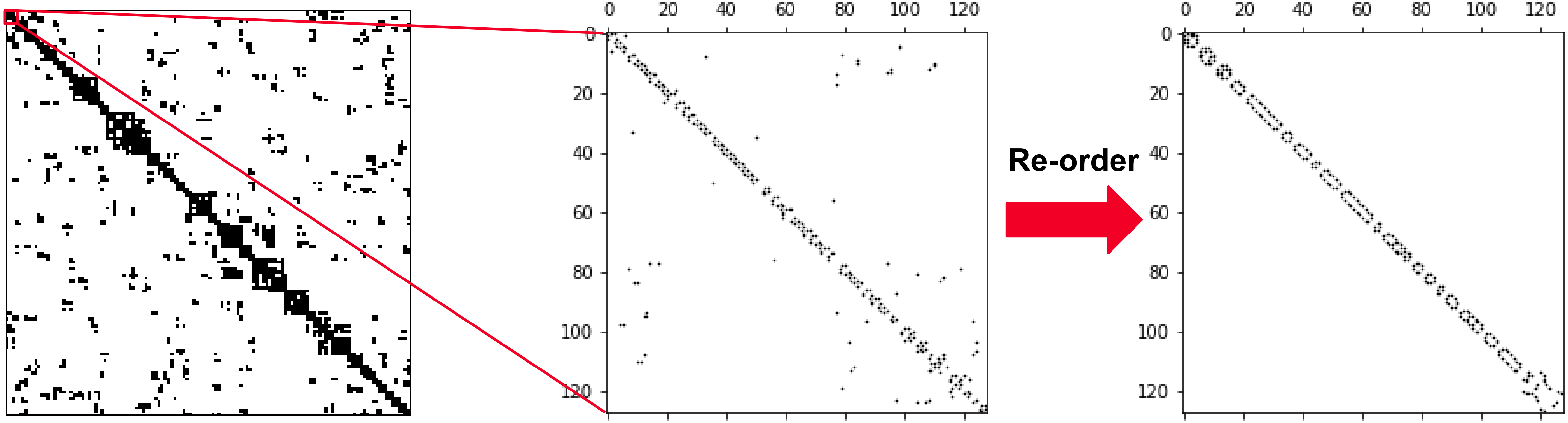
Performance Evaluation

Machine	Details
GPU	Tesla V100-SXM2 (80 SMs, 16GB Global Memory, 640 Tensor Cores, 128 KB L1 cache, 6 MB L2 cache), CUDA version 9.2.88

Dataset

Real symmetric sparse matrix – roadNet-CA

- # of rows and columns = 1,971,281
- Total NNZ = 5,533,214



Re-order

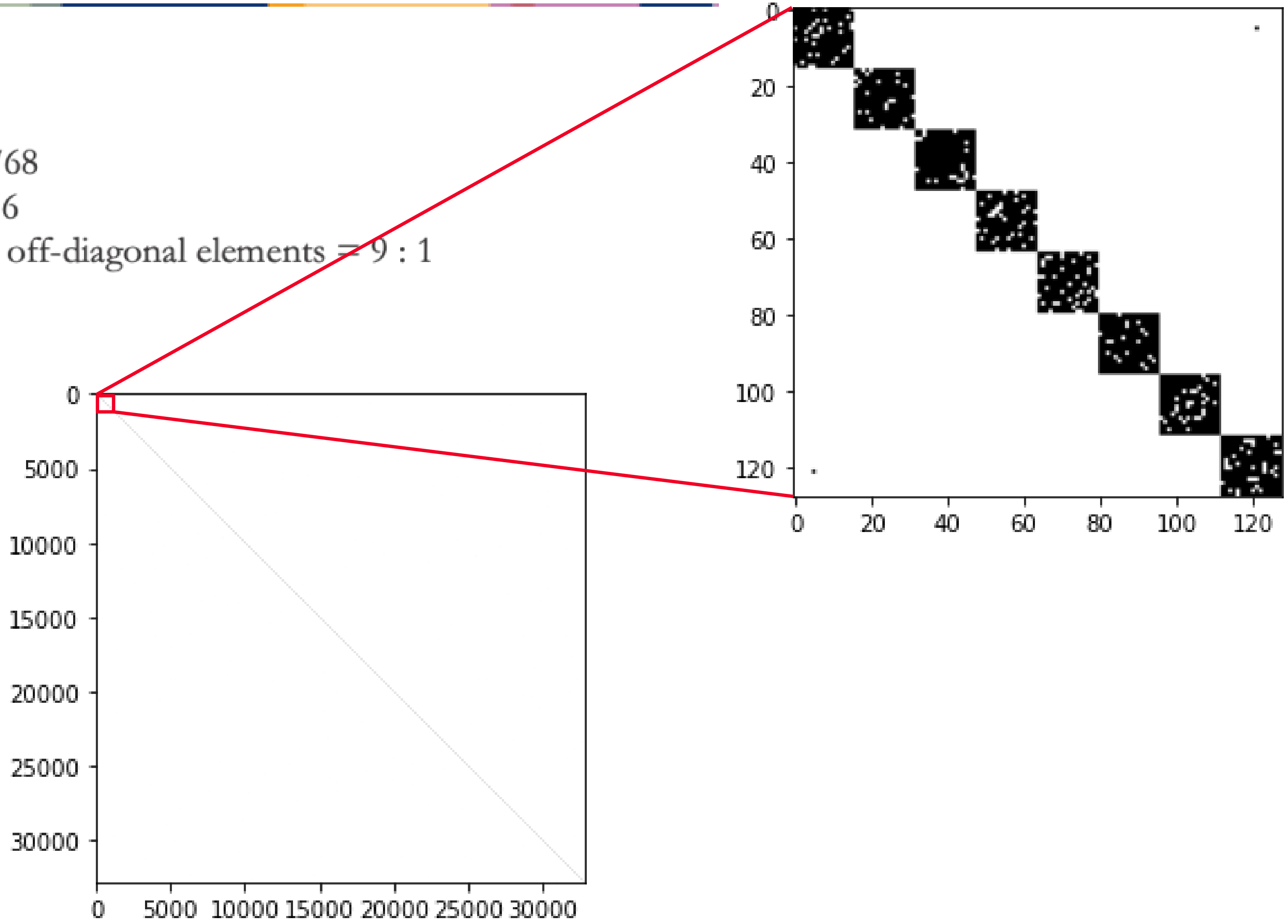
Original sparse matrix

Permuted sparse matrix by RCM

Dataset

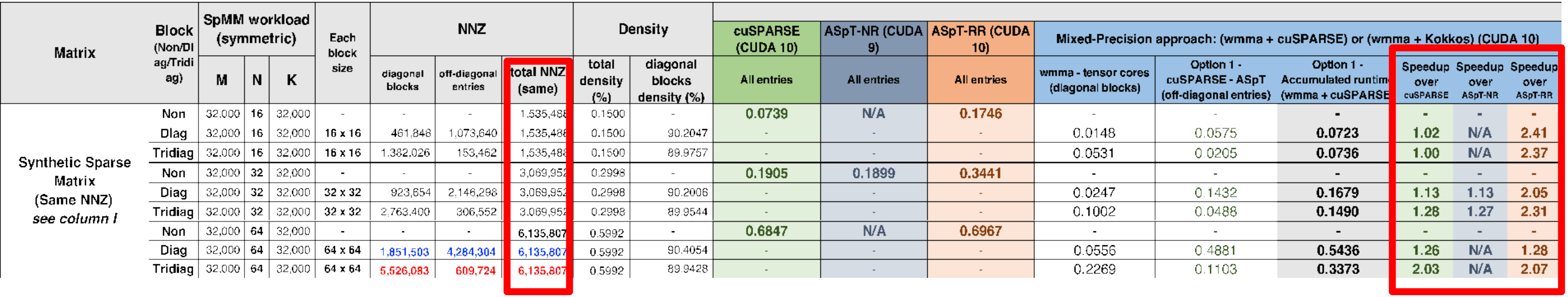
Synthetic RCM-like sparse matrix

- Size of dimensions M and K = 32,768
- Size of each diagonal block = 16×16
- NNZ of diagonal blocks : NNZ of off-diagonal elements = 9 : 1

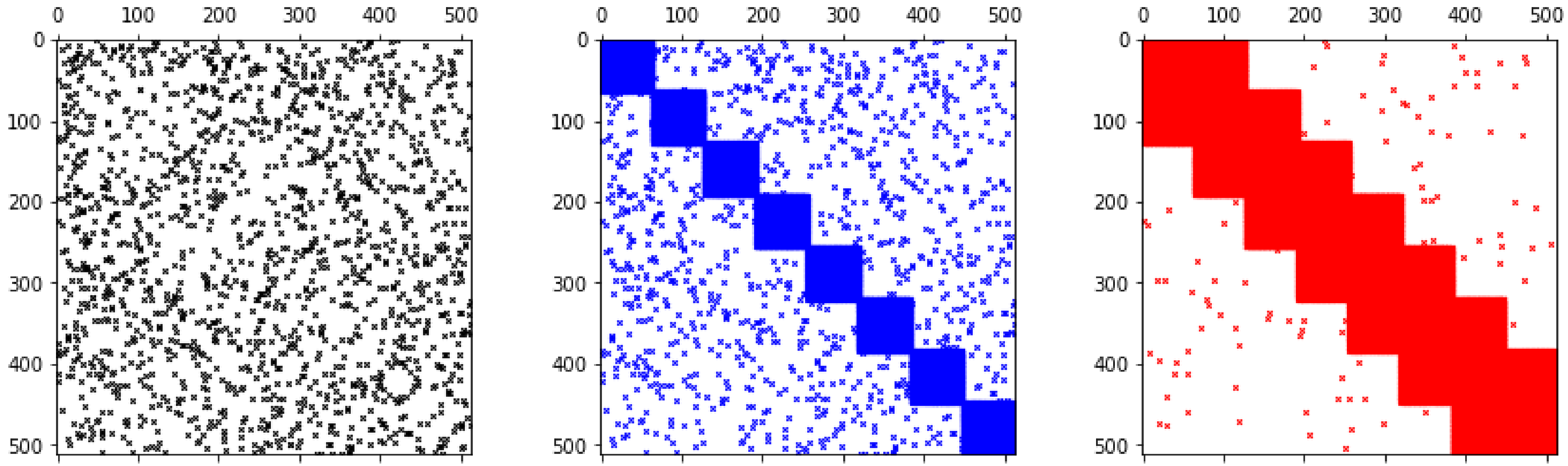


diagonal blocks + off-diagonal entries

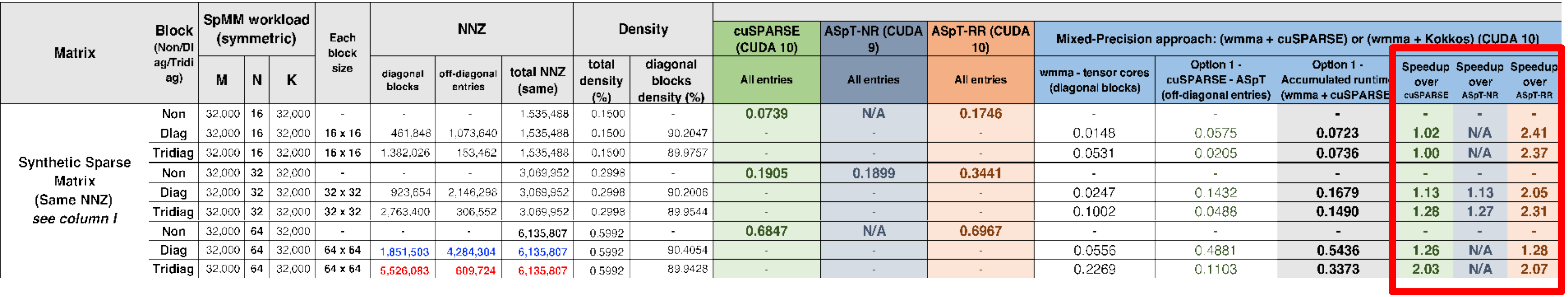
Synthetic case 1



Matrix	Block (Non/Diag/Tridiag)	SpMM workload (symmetric)			Each block size	NNZ			Density		cuSPARSE (CUDA 10)	ASpT-NR (CUDA 9)	ASpT-RR (CUDA 10)	Mixed-Precision approach: (wmma + cuSPARSE) or (wmma + Kokkos) (CUDA 10)					
		M	N	K		diagonal blocks	off-diagonal entries	total NNZ (same)	total density (%)	diagonal blocks density (%)	All entries	All entries	All entries	wmma - tensor cores (diagonal blocks)	Option 1 - cuSPARSE - ASpT (off-diagonal entries)	Option 1 - Accumulated runtim (wmma + cuSPARSE	Speedup over cuSPARSE	Speedup over ASpT-NR	Speedup over ASpT-RR
Synthetic Sparse Matrix (Same NNZ) *see column I*	Non	32,000	16	32,000	-	-	-	1,535,488	0.1500	-	0.0739	N/A	0.1746	-	-	-	-	-	-
	Diag	32,000	16	32,000	16 x 16	461,848	1,073,640	1,535,488	0.1500	90.2047	-	-	-	0.0148	0.0575	0.0723	1.02	N/A	2.41
	Tridiag	32,000	16	32,000	16 x 16	1,382,026	153,462	1,535,488	0.1500	89.9757	-	-	-	0.0531	0.0205	0.0736	1.00	N/A	2.37
	Non	32,000	32	32,000	-	-	-	3,069,952	0.2998	-	0.1905	0.1899	0.3441	-	-	-	-	-	-
	Diag	32,000	32	32,000	32 x 32	923,654	2,146,298	3,069,952	0.2998	90.2006	-	-	-	0.0247	0.1432	0.1679	1.13	1.13	2.05
	Tridiag	32,000	32	32,000	32 x 32	2,763,400	306,552	3,069,952	0.2998	89.9544	-	-	-	0.1002	0.0488	0.1490	1.28	1.27	2.31
	Non	32,000	64	32,000	-	-	-	6,135,807	0.5992	-	0.6847	N/A	0.6967	-	-	-	-	-	-
	Diag	32,000	64	32,000	64 x 64	1,851,503	4,284,304	6,135,807	0.5992	90.4054	-	-	-	0.0556	0.4881	0.5436	1.26	N/A	1.28
	Tridiag	32,000	64	32,000	64 x 64	5,526,083	609,724	6,135,807	0.5992	89.9428	-	-	-	0.2269	0.1103	0.3373	2.03	N/A	2.07



Synthetic matrices



Matrix	Block (Non/Diag/Tridiag)	SpMM workload (symmetric)			Each block size	NNZ			Density		cuSPARSE (CUDA 10)	ASpT-NR (CUDA 9)	ASpT-RR (CUDA 10)	Mixed-Precision approach: (wmma + cuSPARSE) or (wmma + Kokkos) (CUDA 10)					
		M	N	K		diagonal blocks	off-diagonal entries	total NNZ (same)	total density (%)	diagonal blocks density (%)	All entries	All entries	All entries	wmma - tensor cores (diagonal blocks)	Option 1 - cuSPARSE - ASpT (off-diagonal entries)	Option 1 - Accumulated runtim (wmma + cuSPARSE	Speedup over cuSPARSE	Speedup over ASpT-NR	Speedup over ASpT-RR
Synthetic Sparse Matrix (Same NNZ) *see column I*	Non	32,000	16	32,000	-	-	-	1,535,488	0.1500	-	0.0739	N/A	0.1746	-	-	-	-	-	-
	Diag	32,000	16	32,000	16 x 16	461,848	1,073,640	1,535,488	0.1500	90.2047	-	-	-	0.0148	0.0575	0.0723	1.02	N/A	2.41
	Tridiag	32,000	16	32,000	16 x 16	1,382,026	153,462	1,535,488	0.1500	89.9757	-	-	-	0.0531	0.0205	0.0736	1.00	N/A	2.37
	Non	32,000	32	32,000	-	-	-	3,069,952	0.2998	-	0.1905	0.1899	0.3441	-	-	-	-	-	-
	Diag	32,000	32	32,000	32 x 32	923,654	2,146,298	3,069,952	0.2998	90.2006	-	-	-	0.0247	0.1432	0.1679	1.13	1.13	2.05
	Tridiag	32,000	32	32,000	32 x 32	2,763,400	306,552	3,069,952	0.2998	89.9544	-	-	-	0.1002	0.0488	0.1490	1.28	1.27	2.31
	Non	32,000	64	32,000	-	-	-	6,135,807	0.5992	-	0.6847	N/A	0.6967	-	-	-	-	-	-
	Diag	32,000	64	32,000	64 x 64	1,851,503	4,284,304	6,135,807	0.5992	90.4054	-	-	-	0.0556	0.4881	0.5436	1.26	N/A	1.28
	Tridiag	32,000	64	32,000	64 x 64	5,526,083	609,724	6,135,807	0.5992	89.9428	-	-	-	0.2269	0.1103	0.3373	2.03	N/A	2.07

Keep the number of non-zeros constant, increase the non-zeros within the dense blocks

- As expected, the more non-zeros within the dense block better the performance

This is comparing cuSparse with cuSparse + dense blocks on GPUs

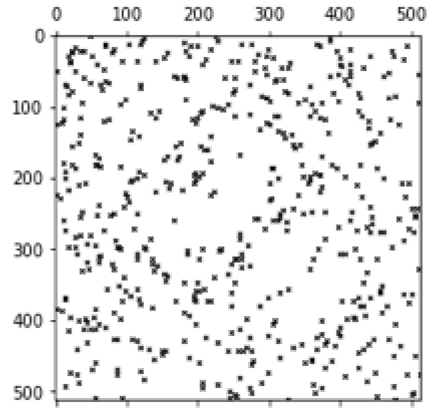
We assumed diagonal or tridiagonal blocks are mostly dense (may not happen in real problems)

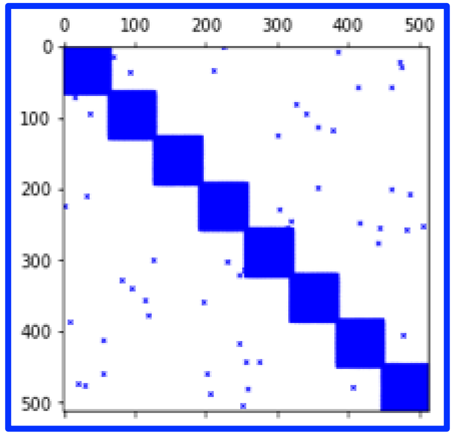
Explicit tiled implementations that do not use tensor cores cannot benefit from these use cases

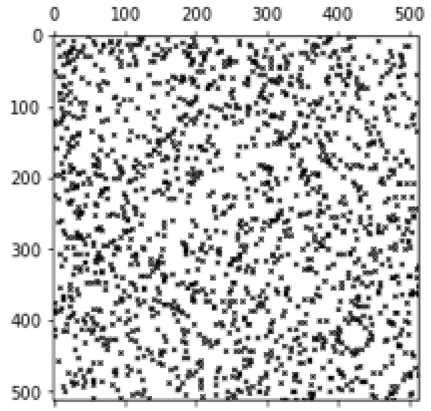
Validates the intuition that using the hardware will be beneficial even for sparse problems if dense blocks can be identified

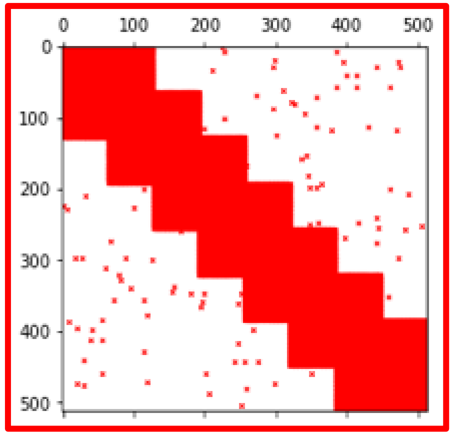
Synthetic Matrices Use Case - 2

Matrix	Block (Non/Diag/Tridiag)	SpMM workload (symmetric)			Each block size	NNZ			Density		cuSPARSE (CUDA 10)	ASpT-NR (CUDA 9)	ASpT-RR (CUDA 10)	Mixed-Precision approach: (wmma + cuSPARSE) or (wmma + Kokkos) (CUDA 10)					
		M	N	K		diagonal blocks (~90%)	off-diagonal entries (~10%)	total NNZ (different)	total density (%)	diagonal blocks density (%)	All entries	All entries	All entries	wmma - tensor cores (diagonal blocks)	Option 1 - cuSPARSE - ASpT (off-diagonal entries)	Option 1 - Accumulated runtime (wmma + cuSPARSE)	Speedup over cuSPARSE	Speedup over ASpT-NR	Speedup over ASpT-RR
Synthetic Sparse Matrix (Different NNZ) *see column I*	Non	32,000	16	32,000	-	-	-	512,479	0.0500	-	0.0375	N/A	0.0641	-	-	-	-	-	-
	Diag	32,000	16	32,000	16 x 16	461,335	51,144	512,479	0.0500	90.1045	-	-	-	0.0145	0.0175	0.0320	1.17	N/A	2.00
	Non	32,000	16	32,000	-	-	-	1,535,198	0.1499	-	0.0730	N/A	0.1742	-	-	-	-	-	-
	Tridiag	32,000	16	32,000	16 x 16	1,382,024	153,174	1,535,198	0.1499	89.9755	-	-	-	0.0525	0.0198	0.0723	1.01	N/A	2.41
	Non	32,000	32	32,000	-	-	-	1,023,975	0.1000	-	0.0762	0.0795	0.1185	-	-	-	-	-	-
	Diag	32,000	32	32,000	32 x 32	921,646	102,330	1,023,976	0.1000	90.0045	-	-	-	0.0247	0.0412	0.0659	1.16	1.21	1.80
	Non	32,000	32	32,000	-	-	-	3,069,458	0.2998	-	0.1891	0.1903	0.3446	-	-	-	-	-	-
	Tridiag	32,000	32	32,000	32 x 32	2,763,400	306,058	3,069,458	0.2998	89.9544	-	-	-	0.1006	0.0481	0.1487	1.27	1.28	2.32
	Non	32,000	64	32,000	-	-	-	2,047,815	0.2000	-	0.2602	N/A	0.234	-	-	-	-	-	-
	Diag	32,000	64	32,000	64 x 64	1,843,268	204,548	2,047,816	0.2000	90.0033	-	-	-	0.0558	0.0876	0.1434	1.82	N/A	1.63
	Non	32,000	64	32,000	-	-	-	6,138,573	0.5995	-	0.6848	N/A	0.6949	-	-	-	-	-	-
	Tridiag	32,000	64	32,000	64 x 64	5,526,103	612,470	6,138,573	0.5995	89.9431	-	-	-	0.2226	0.1097	0.3323	2.06	N/A	2.09



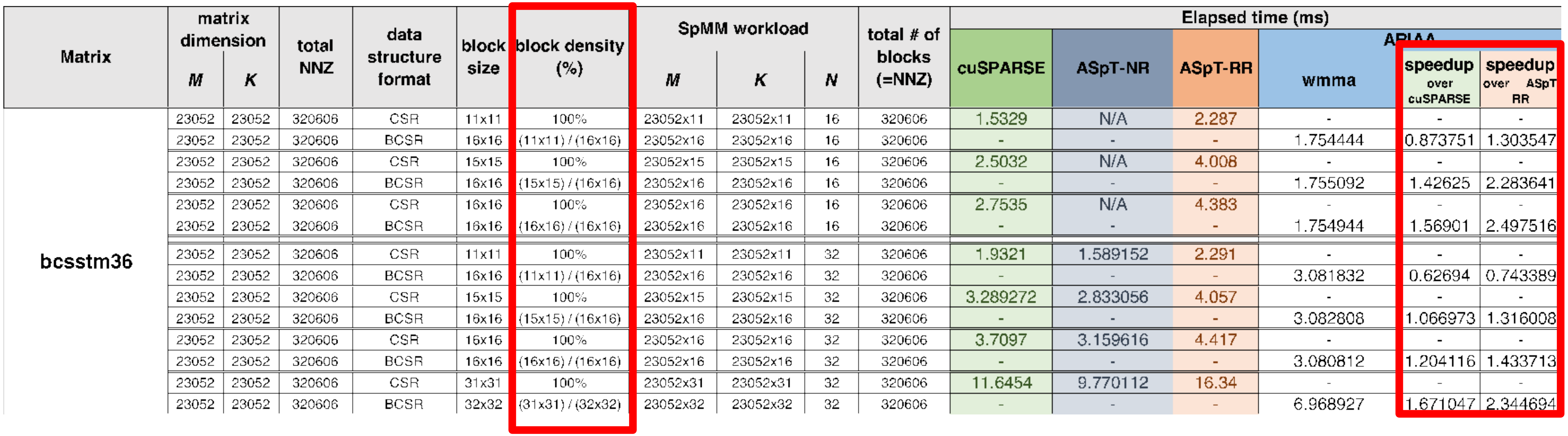




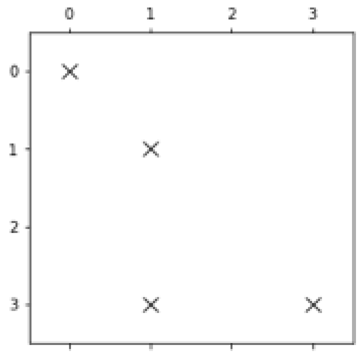


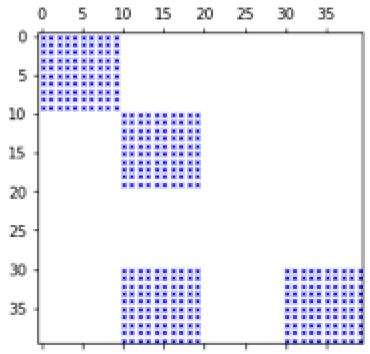
The benefits are better for denser problems of same dimension as expected

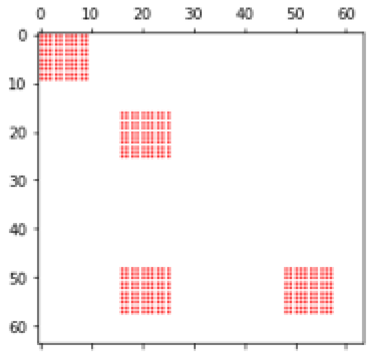
Multiphysics use case



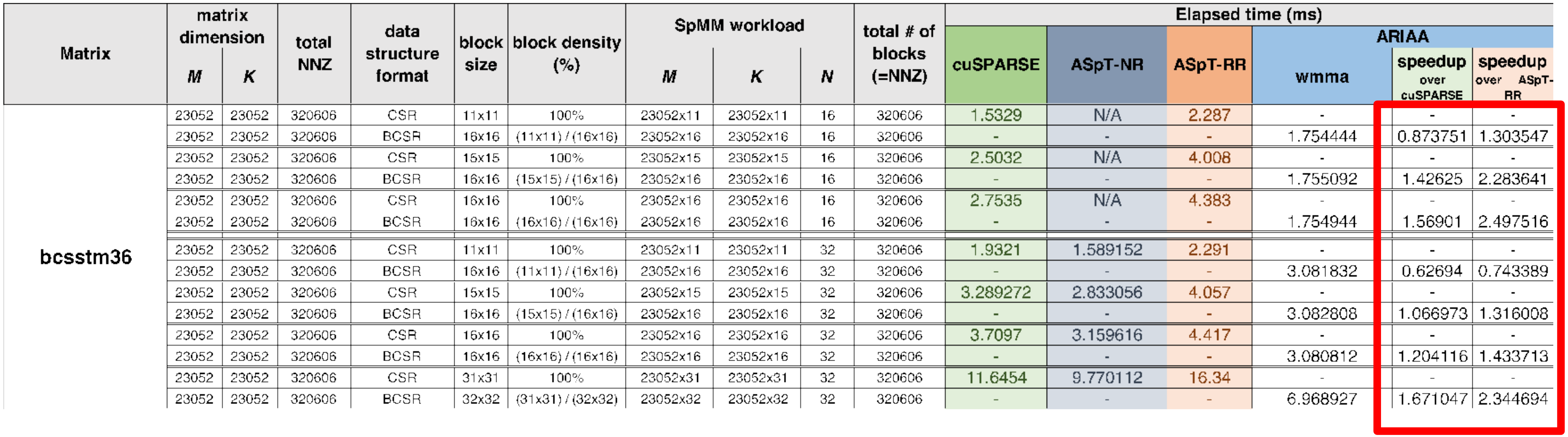
Matrix	matrix dimension		total NNZ	data structure format	block size	block density (%)	SpMM workload			total # of blocks (=NNZ)	Elapsed time (ms)					
														APIAA		
	M	*K*					*M*	*K*	*N*		cuSPARSE	ASpT-NR	ASpT-RR	wmma	speedup over cuSPARSE	speedup over ASpT RR
bcsstm36	23052	23052	320606	CSR	11x11	100%	23052x11	23052x11	16	320606	1.5329	N/A	2.287	-	-	-
	23052	23052	320606	BCSR	16x16	(11x11) / (16x16)	23052x16	23052x16	16	320606	-	-	-	1.754444	0.873751	1.303547
	23052	23052	320606	CSR	15x15	100%	23052x15	23052x15	16	320606	2.5032	N/A	4.008	-	-	-
	23052	23052	320606	BCSR	16x16	(15x15) / (16x16)	23052x16	23052x16	16	320606	-	-	-	1.755092	1.42625	2.283641
	23052	23052	320606	CSR	16x16	100%	23052x16	23052x16	16	320606	2.7535	N/A	4.383	-	-	-
	23052	23052	320606	BCSR	16x16	(16x16) / (16x16)	23052x16	23052x16	16	320606	-	-	-	1.754944	1.56901	2.497516
	23052	23052	320606	CSR	11x11	100%	23052x11	23052x11	32	320606	1.9321	1.589152	2.291	-	-	-
	23052	23052	320606	BCSR	16x16	(11x11) / (16x16)	23052x16	23052x16	32	320606	-	-	-	3.081832	0.62694	0.743389
	23052	23052	320606	CSR	15x15	100%	23052x15	23052x15	32	320606	3.289272	2.833056	4.057	-	-	-
	23052	23052	320606	BCSR	16x16	(15x15) / (16x16)	23052x16	23052x16	32	320606	-	-	-	3.082808	1.066973	1.316008
	23052	23052	320606	CSR	16x16	100%	23052x16	23052x16	32	320606	3.7097	3.159616	4.417	-	-	-
	23052	23052	320606	BCSR	16x16	(16x16) / (16x16)	23052x16	23052x16	32	320606	-	-	-	3.080812	1.204116	1.433713
	23052	23052	320606	CSR	31x31	100%	23052x31	23052x31	32	320606	11.6454	9.770112	16.34	-	-	-
	23052	23052	320606	BCSR	32x32	(31x31) / (32x32)	23052x32	23052x32	32	320606	-	-	-	6.968927	1.671047	2.344694







Multiphysics case



Matrix	matrix dimension		total NNZ	data structure format	block size	block density (%)	SpMM workload			total # of blocks (=NNZ)	Elapsed time (ms)					
											cuSPARSE	ASpT-NR	ASpT-RR	ARIAA		
	M	*K*					*M*	*K*	*N*					wmma	speedup over cuSPARSE	speedup over ASpT-RR
bcsstm36	23052	23052	320606	CSR	11x11	100%	23052x11	23052x11	16	320606	1.5329	N/A	2.287	-	-	-
	23052	23052	320606	BCSR	16x16	(11x11) / (16x16)	23052x16	23052x16	16	320606	-	-	-	1.754444	0.873751	1.303547
	23052	23052	320606	CSR	15x15	100%	23052x15	23052x15	16	320606	2.5032	N/A	4.008	-	-	-
	23052	23052	320606	BCSR	16x16	(15x15) / (16x16)	23052x16	23052x16	16	320606	-	-	-	1.755092	1.42625	2.283641
	23052	23052	320606	CSR	16x16	100%	23052x16	23052x16	16	320606	2.7535	N/A	4.383	-	-	-
	23052	23052	320606	BCSR	16x16	(16x16) / (16x16)	23052x16	23052x16	16	320606	-	-	-	1.754944	1.56901	2.497516
	23052	23052	320606	CSR	11x11	100%	23052x11	23052x11	32	320606	1.9321	1.589152	2.291	-	-	-
	23052	23052	320606	BCSR	16x16	(11x11) / (16x16)	23052x16	23052x16	32	320606	-	-	-	3.081832	0.62694	0.743389
	23052	23052	320606	CSR	15x15	100%	23052x15	23052x15	32	320606	3.289272	2.833056	4.057	-	-	-
	23052	23052	320606	BCSR	16x16	(15x15) / (16x16)	23052x16	23052x16	32	320606	-	-	-	3.082808	1.066973	1.316008
	23052	23052	320606	CSR	16x16	100%	23052x16	23052x16	32	320606	3.7097	3.159616	4.417	-	-	-
	23052	23052	320606	BCSR	16x16	(16x16) / (16x16)	23052x16	23052x16	32	320606	-	-	-	3.080812	1.204116	1.433713
	23052	23052	320606	CSR	31x31	100%	23052x31	23052x31	32	320606	11.6454	9.770112	16.34	-	-	-
	23052	23052	320606	BCSR	32x32	(31x31) / (32x32)	23052x32	23052x32	32	320606	-	-	-	6.968927	1.671047	2.344694

Synthetic Multiphysics matrices but with block sizes typically expected from physics codes

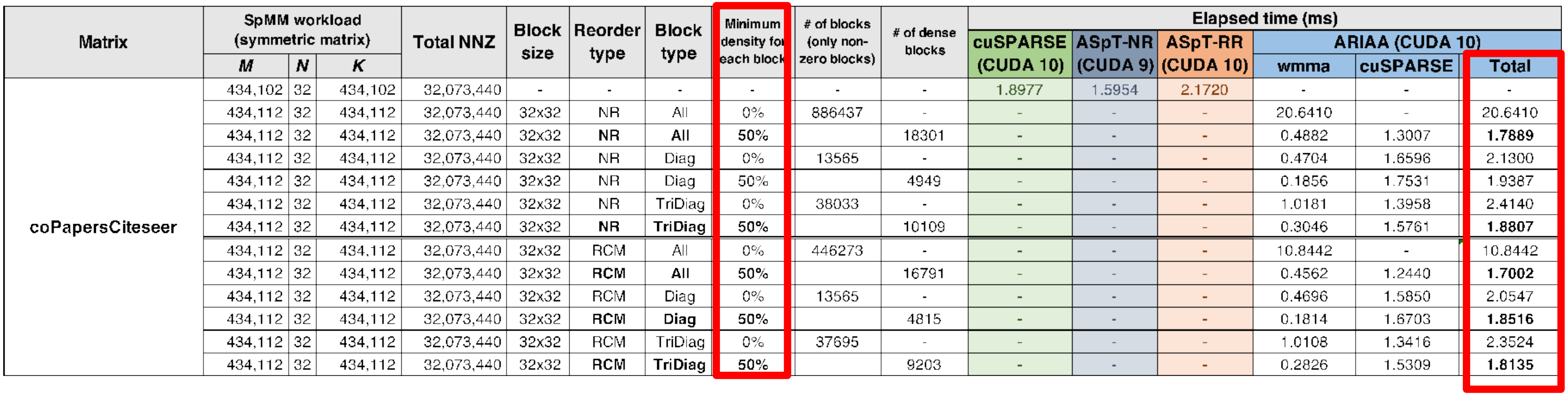
- 11x11, 15x15, 31x31

Pad the block sizes to match the hardware expectations and add more flops and memory

The approach helps when the padding is small enough

Validates the intuition that this is a good approach for multiphysics codes

General case



Matrix	SpMM workload (symmetric matrix)			Total NNZ	Block size	Reorder type	Block type	Minimum density for each block	# of blocks (only non-zero blocks)	# of dense blocks	Elapsed time (ms)					
											cuSPARSE (CUDA 10)	ASpT-NR (CUDA 9)	ASpT-RR (CUDA 10)	ARIAA (CUDA 10)		
	M	*N*	*K*											wmma	cuSPARSE	Total
coPapersCiteseer	434,102	32	434,102	32,073,440	-	-	-	-	-	-	1.8977	1.5954	2.1720	-	-	-
	434,112	32	434,112	32,073,440	32x32	NR	All	0%	886437	-	-	-	-	20.6410	-	20.6410
	434,112	32	434,112	32,073,440	32x32	**NR**	**All**	**50%**		18301	-	-	-	0.4882	1.3007	**1.7889**
	434,112	32	434,112	32,073,440	32x32	NR	Diag	0%	13565	-	-	-	-	0.4704	1.6596	2.1300
	434,112	32	434,112	32,073,440	32x32	NR	Diag	50%		4949	-	-	-	0.1856	1.7531	1.9387
	434,112	32	434,112	32,073,440	32x32	NR	TriDiag	0%	38033	-	-	-	-	1.0181	1.3958	2.4140
	434,112	32	434,112	32,073,440	32x32	**NR**	**TriDiag**	**50%**		10109	-	-	-	0.3046	1.5761	**1.8807**
	434,112	32	434,112	32,073,440	32x32	RCM	All	0%	446273	-	-	-	-	10.8442	-	10.8442
	434,112	32	434,112	32,073,440	32x32	**RCM**	**All**	**50%**		16791	-	-	-	0.4562	1.2440	**1.7002**
	434,112	32	434,112	32,073,440	32x32	RCM	Diag	0%	13565	-	-	-	-	0.4696	1.5850	2.0547
	434,112	32	434,112	32,073,440	32x32	**RCM**	**Diag**	**50%**		4815	-	-	-	0.1814	1.6703	**1.8516**
	434,112	32	434,112	32,073,440	32x32	RCM	TriDiag	0%	37695	-	-	-	-	1.0108	1.3416	2.3524
	434,112	32	434,112	32,073,440	32x32	**RCM**	**TriDiag**	**50%**		9203	-	-	-	0.2826	1.5309	**1.8135**

Form blocks based on a minimum density for blocks (50% shown)

The performance improves as the number of blocks that meet the minimum criteria increases

Still this is only on par with a standard SpMM

Currently two separate phases in calls ? Can we overlap the sparse portion with the dense portion ?

Can we use variable size blocks ? Can we use the tiling strategies with the tensor

Outline

Utilizing low-precision hardware for SpMV

- Use existing hardware (GPUs) to evaluate algorithmic approaches
- Can we use current low-precision hardware for sparse-matrix methods?

Simulations of future hardware for GEMMs

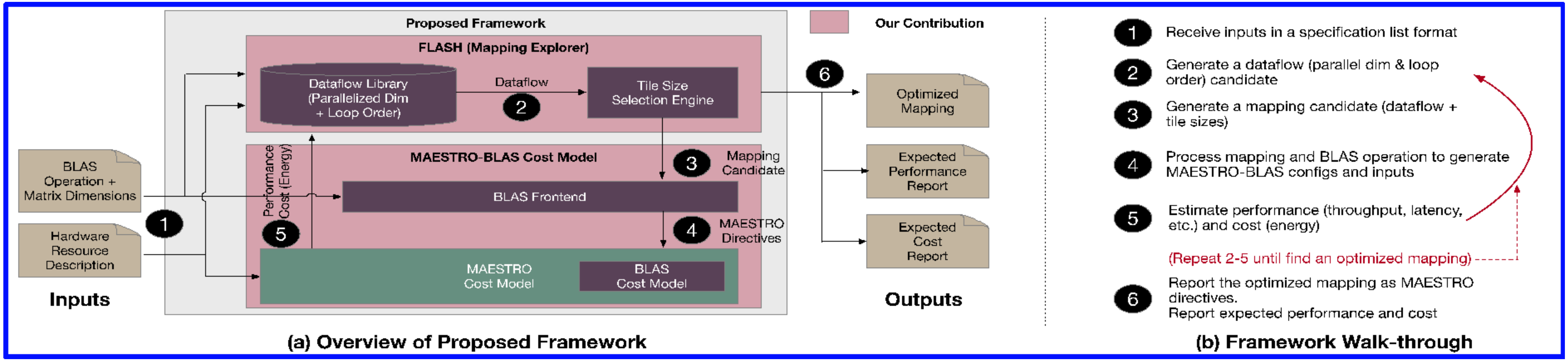
- How do we generalize to different block sizes?
- How can we study different GEMM sizes on different types of accelerators?
- Can we study (using simulations) other hardware for GEMMS?

ML / GEMM Accelerators

ML Accelerators

- Several ML accelerators are available or proposed (TPUs, MAERI, Eyeriss, ShiDiaNano)
- Many of them support more general matrix and vector sizes
- They use different variations for GEMM (input / output / weight stationary)
- Can we evaluate how the accelerators for different workload
- Our Solution: Use an analytical model (MAESTRO-BLAS) to evaluate the accelerators while choosing the best mapping for each of the accelerator using a mapping tool (FLASH)
- *Evaluating Spatial Accelerator Architectures with Tiled Matrix-Matrix Multiplication*, *Gordon E. Moon, Hyoukjun Kwon, Geonhwa Jeong, Prasanth*

Mapping Space Exploration of BLAS Kernels on Dataflow Accelerators



(a) Overview of Proposed Framework

1. Receive inputs in a specification list format
2. Generate a dataflow (parallel dim & loop order) candidate
3. Generate a mapping candidate (dataflow + tile sizes)
4. Process mapping and BLAS operation to generate MAESTRO-BLAS configs and inputs
5. Estimate performance (throughput, latency, etc.) and cost (energy)

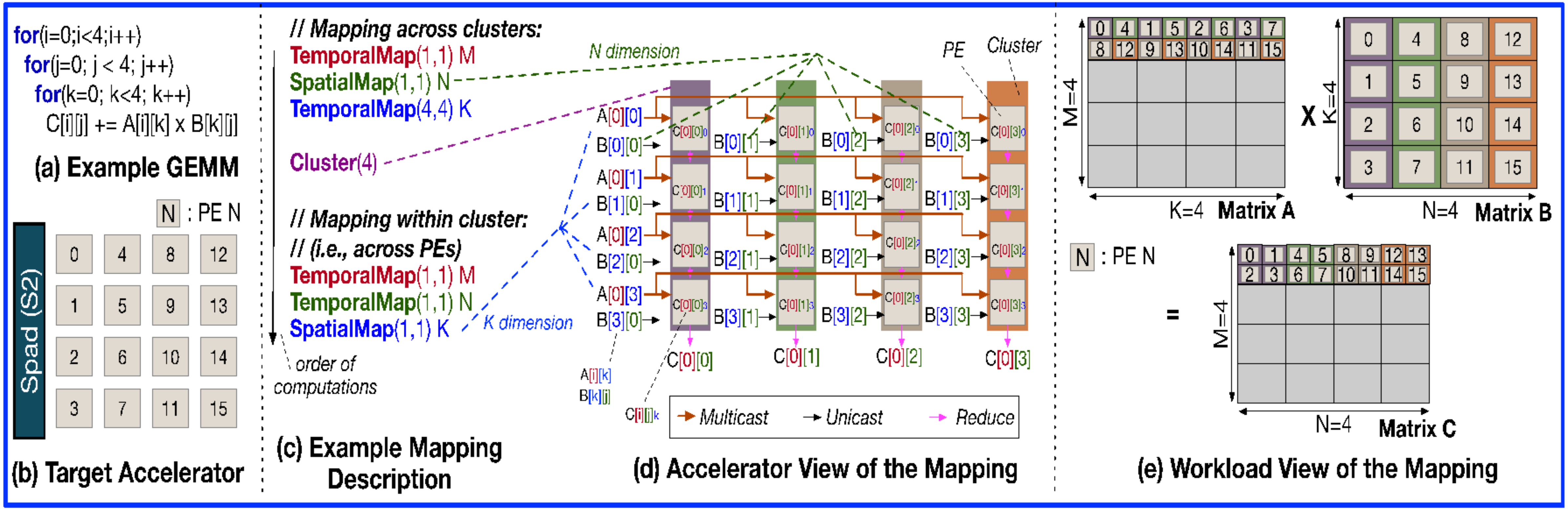
(Repeat 2-5 until find an optimized mapping)

6. Report the optimized mapping as MAESTRO directives. Report expected performance and cost

(b) Framework Walk-through

- We developed a detailed analytical model, **MAESTRO-BLAS**, to evaluate several dataflows inspired by popular ML dataflow accelerators such as **TPU (Google), NVDLA (NVIDIA), Eyeriss (MIT), MAERI (Georgia Tech) and ShiDianNao (China)**
    - MAESTRO-BLAS allows us to **co-design GEMM architectures and algorithms** understand how current ML accelerators can work for Computational Science and Engineering use cases
- We developed a mapping explorer for BLAS called FLASH to evaluate mappings and tile-sizes that are efficient for BLAS kernels on dataflow accelerators

Mapping Space Exploration of BLAS Kernels on Dataflow Accelerators



```
for(i=0;i<4;i++)
  for(j=0; j < 4; j++)
    for(k=0; k<4; k++)
      C[i][j] += A[i][k] x B[k][j]
```

(a) Example GEMM

(b) Target Accelerator

```
// Mapping across clusters:
TemporalMap(1,1) M
SpatialMap(1,1) N
TemporalMap(4,4) K

Cluster(4)

// Mapping within cluster:
// (i.e., across PEs)
TemporalMap(1,1) M
TemporalMap(1,1) N
SpatialMap(1,1) K
```

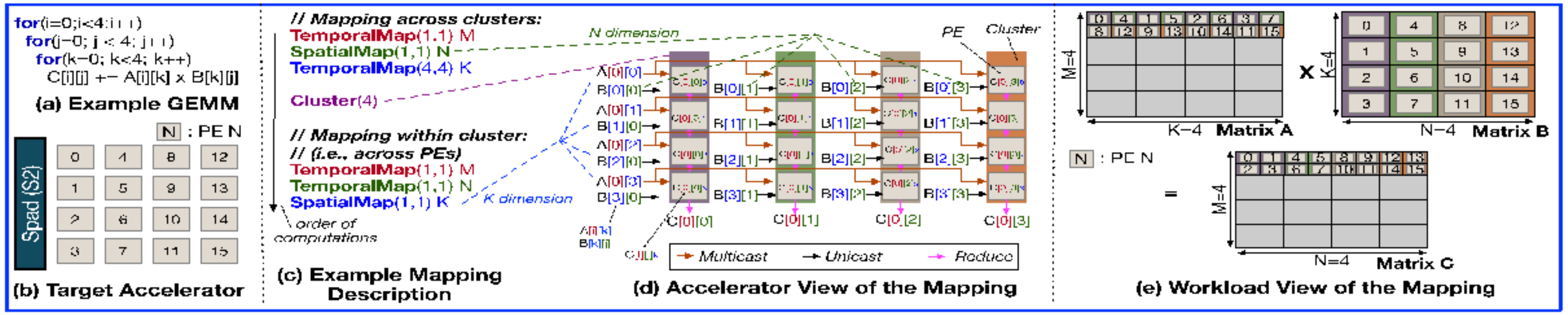
(c) Example Mapping Description

(d) Accelerator View of the Mapping

(e) Workload View of the Mapping

- ➢ We can study several different mappings of GEMM variants
  - ➢ Tensor Core mapping is one option here
- ➢ Completed comparison of different GEMM algorithms and dataflows to demonstrate differences in runtime, energy and throughput for an edge accelerators and a cloud/HPC accelerator for different GEMM workloads

Mapping Space Exploration of BLAS Kernels on Dataflow Accelerators



```
for(i=0;i<4;i++)
  for(j=0; j < 4; j++)
    for(k=0; k<4; k++)
      C[i][j] += A[i][k] x B[k][j]
```

(a) Example GEMM

(b) Target Accelerator

```
// Mapping across clusters:
TemporalMap(1,1) M
SpatialMap(1,1) N
TemporalMap(4,4) K

Cluster(4)

// Mapping within cluster:
// (i.e., across PEs)
TemporalMap(1,1) M
TemporalMap(1,1) N
SpatialMap(1,1) K
```

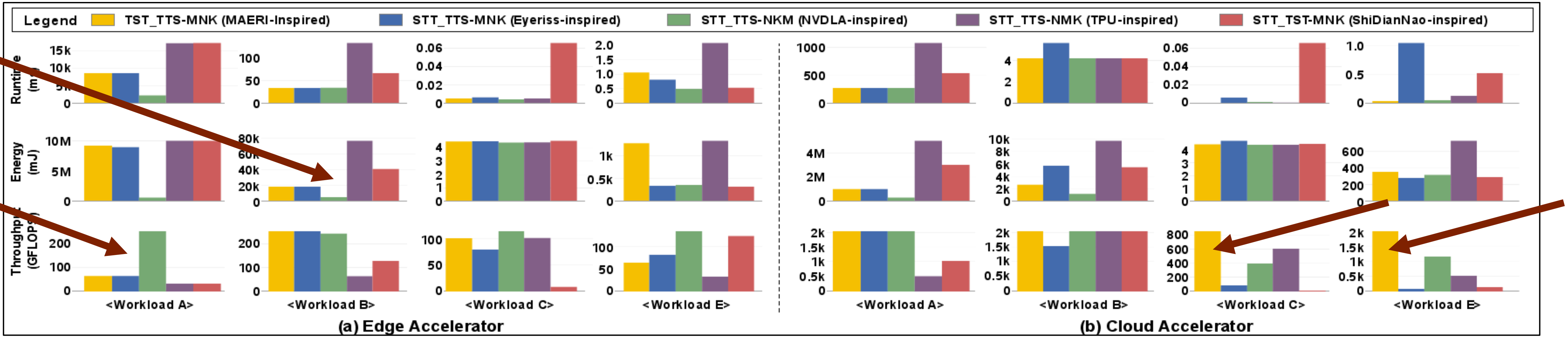
(c) Example Mapping Description

(d) Accelerator View of the Mapping

(e) Workload View of the Mapping

MAESTRO-BLAS Cost Model

Matrix Dimension	Workload ID					
	I	II	III	IV	V	VI
M	8192	1024	8	8	8192	512
N	8192	1024	8	8192	8	256
K	8192	8192	8192	1024	1024	256
GFLOPs	549.8	8.59	0.001	0.067	0.067	0.03



(a) Edge Accelerator

(b) Cloud Accelerator

Takeaway: FLASH + MAESTRO-BLAS demonstrate that the algorithm of choice (loop order), tile size, and cluster size vary widely for ML accelerators.

Summary

Utilizing low-precision hardware for SpMM

- Use of tensor cores are beneficial for Multiphysics use cases when block sizes are close to hardware expected sizes
- Better reordering/tiling strategies to find dense problems could assist in improving performance for general sparse use case
- Overlapping sparse and dense compute, hybrid approaches with tiling and tensor cores could help as well.

Simulations of future hardware for GEMMs

- Several hardware choices are becoming available for low precision GEMMs and even SpMMs
- Based on simulations there is potential to use at least some of these hardware for sparse computations

Thanks! Questions?

Exceptional service in the national interest

Sandia National Laboratories