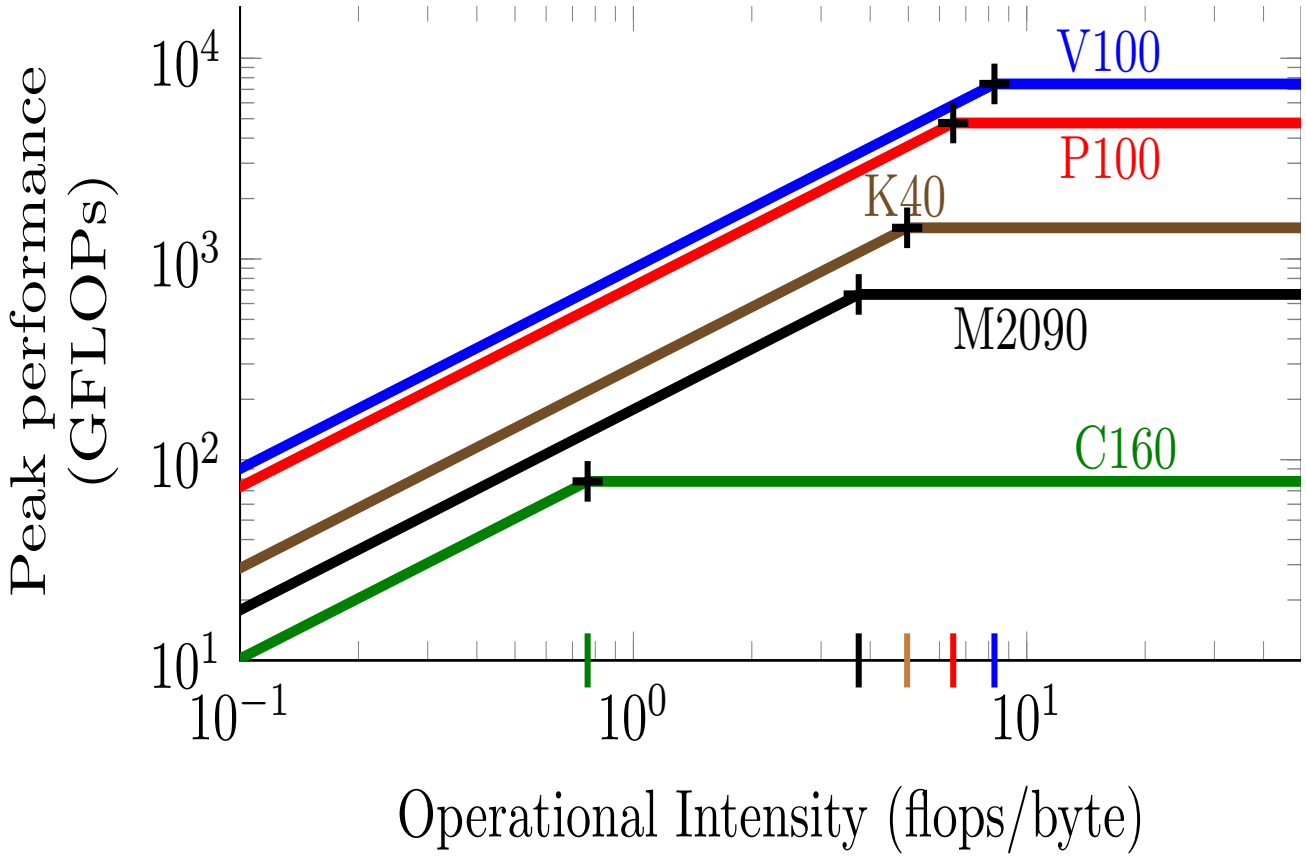
Data Locality Optimization for Tensor Computations

P. (Saday) Sadayappan
Ohio State University

MIT Sparse Tensor Workshop
January 26, 2019

Data Reuse is Increasingly Important



- Nvidia GPUs over 5 generations: Fermi, Maxwell, Kepler, Pascal, Volta
- Peak GFLOPs and Peak Mem BW have both increased
- But machine balance (Peak_GFLOPs/Peak_BW) has steadily risen => more and more constrained by data movement

Data Locality Optimization: Dense Tensor Contractions

$$C_{ijkl} = \sum_{mn} A_{imkn} \cdot B_{jnlm}$$

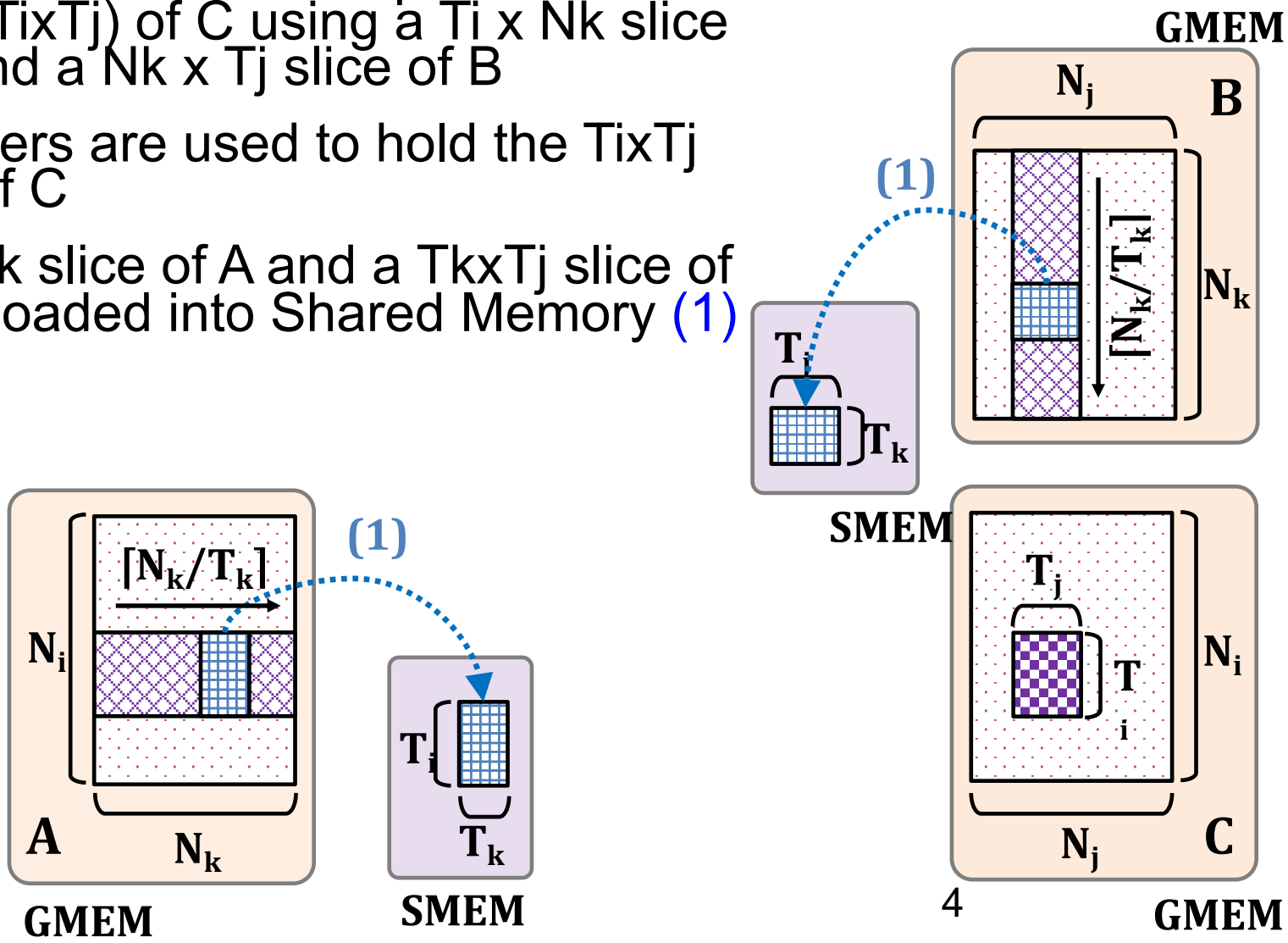
```
for (i=0; i<N; i++)
 for (j=0; j<N; j++)
  for (k=0; k<N; k++)
   for (l=0; l<N; l++)
    for (m=0; m<N; m++)
     for (n=0; n<N; n++)
      C[i][j][k][l] += A[i][m][k][n]*B[j][n][l][m];
```

- Tensor contraction is high-dimension analog of matrix-matrix product
- Each loop index appears in exactly two tensors
  - "Contraction index" appears only in input (rhs) tensors: {m, n}
  - "External index": appears in output tensor and one input tensor: {i, k} {j, l}
- TensorGen project (OSU/PNNL) is developing domain-specific compiler for multi-target (GPU, multi/manycore CPU) optimization of arbitrary tensor contractions
  - Specialized schema for optimized data movement/buffering

Matrix Multiplication Schema

C[i][j] += A[i][k]*B[k][j]

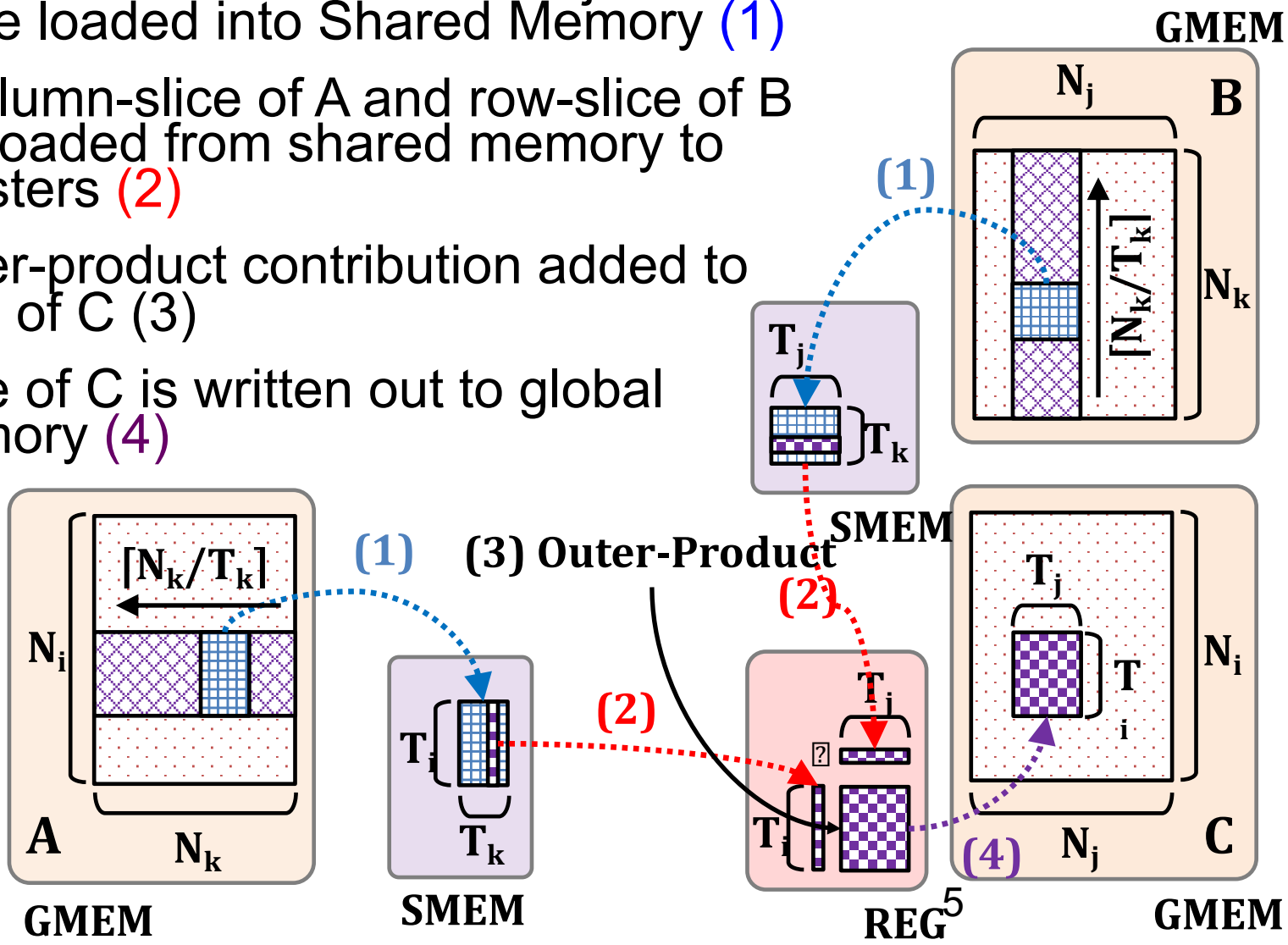
- A 2D thread-block computes a 2D slice (TixTj) of C using a Ti x Nk slice of A and a Nk x Tj slice of B
- Registers are used to hold the TixTj slice of C
- A TixTk slice of A and a TkxTj slice of B are loaded into Shared Memory (1)



Matrix Multiplication Schema

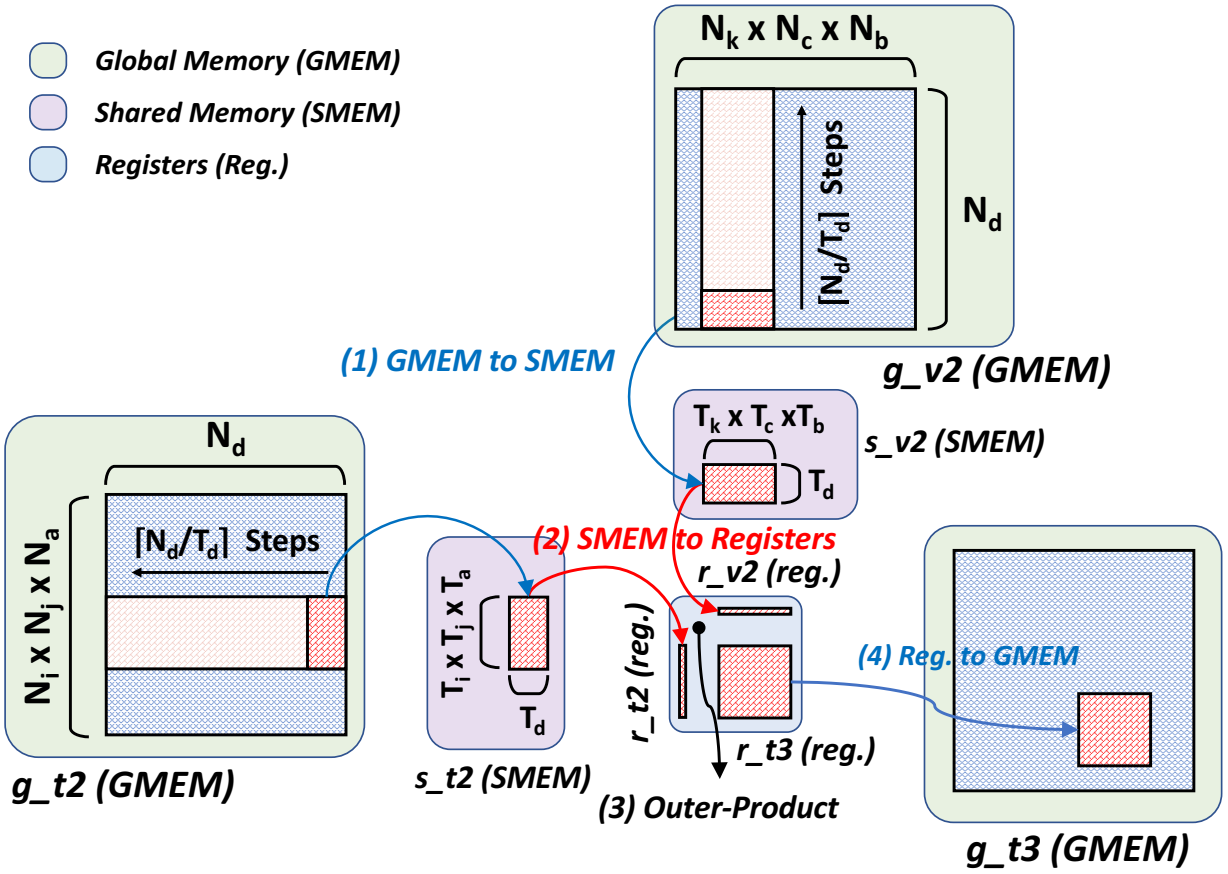
C[i][j] += A[i][k]*B[k][j]

- A TixTk slice of A and a TkxTj slice of B are loaded into Shared Memory (1)
- A column-slice of A and row-slice of B are loaded from shared memory to registers (2)
- Outer-product contribution added to slice of C (3)
- Slice of C is written out to global memory (4)



Generalizing for Arbitrary Tensor Contractions

t3[k, j, i, c, b, a] -= t2[d, a, i, j] * v2[d, k, c, b]

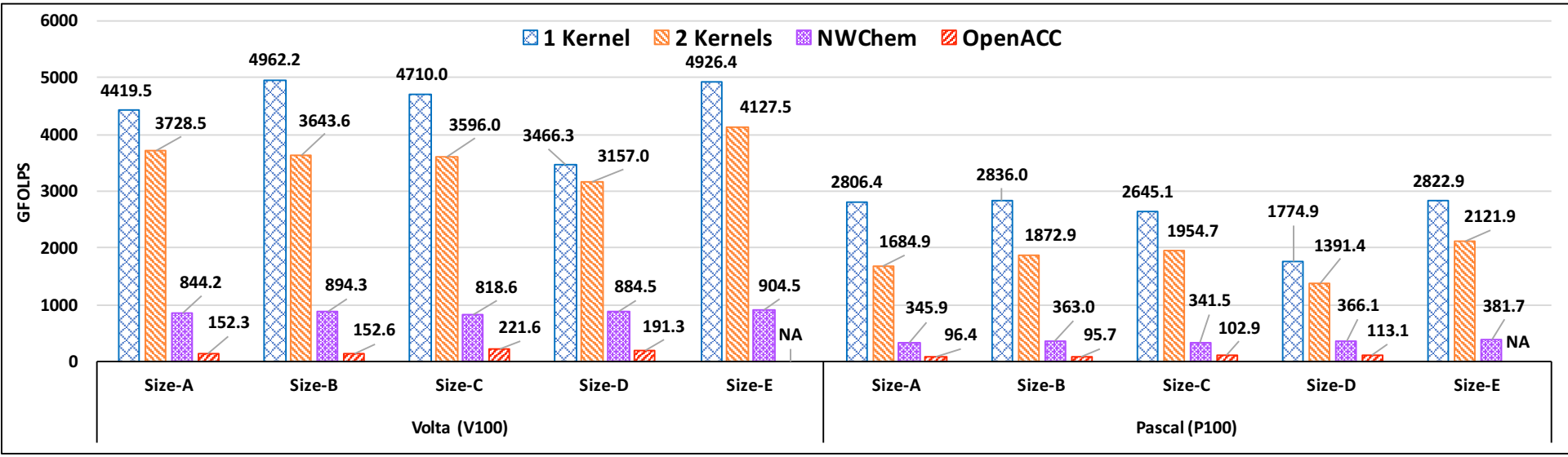


- Use same "outer-product" scheme in registers as dense mat-mult
- Slice of t3 held in register tiles; maximize reuse of data slices of t2 and v2
- Many possible ways to map tensor indexes to registers/threads (space) & explicit loops (time)

CCSD(T) Tensor Contractions in NWChem

sd1_1	$t3[k,j,i,c,b,a] -= t2[l,a,b,i] * v2[k,j,c,l]$
sd1_2	$t3[k,j,i,c,b,a] += t2[l,a,b,j] * v2[k,i,c,l]$
sd1_3	$t3[k,j,i,c,b,a] -= t2[l,a,b,k] * v2[j,i,c,l]$
sd1_4	$t3[k,j,i,c,b,a] -= t2[l,b,c,i] * v2[k,j,a,l]$
sd1_5	$t3[k,j,i,c,b,a] += t2[l,b,c,j] * v2[k,i,a,l]$
sd1_6	$t3[k,j,i,c,b,a] -= t2[l,b,c,k] * v2[j,i,a,l]$
sd1_7	$t3[k,j,i,c,b,a] += t2[l,a,c,i] * v2[k,j,b,l]$
sd1_8	$t3[k,j,i,c,b,a] -= t2[l,a,c,j] * v2[k,i,b,l]$
sd1_9	$t3[k,j,i,c,b,a] += t2[l,a,c,k] * v2[j,i,b,l]$

sd2_1	$t3[k,j,i,c,b,a] -= t2[d,a,i,j] * v2[d,k,c,b]$
sd2_2	$t3[k,j,i,c,b,a] -= t2[d,a,j,k] * v2[d,i,c,b]$
sd2_3	$t3[k,j,i,c,b,a] += t2[d,a,i,k] * v2[d,j,c,b]$
sd2_4	$t3[k,j,i,c,b,a] += t2[d,b,i,j] * v2[d,k,c,a]$
sd2_5	$t3[k,j,i,c,b,a] += t2[d,b,j,k] * v2[d,i,c,a]$
sd2_6	$t3[k,j,i,c,b,a] -= t2[d,b,i,k] * v2[d,j,c,a]$
sd2_7	$t3[k,j,i,c,b,a] -= t2[d,c,i,j] * v2[d,k,b,a]$
sd2_8	$t3[k,j,i,c,b,a] -= t2[d,c,j,k] * v2[d,i,b,a]$
sd2_9	$t3[k,j,i,c,b,a] += t2[d,c,i,k] * v2[d,j,b,a]$



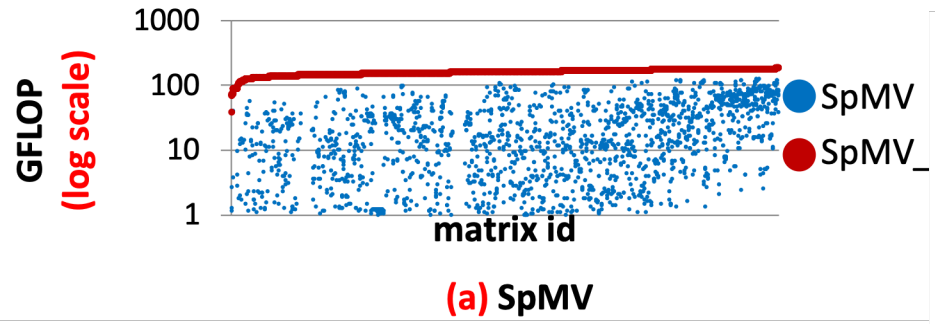
- CCSD(T) is an accurate but extremely compute-intensive method in NWChem
- Newl GPU kernels significantly outperform current GPU code in NWChem
- Code is being incorporated into NWChemEX

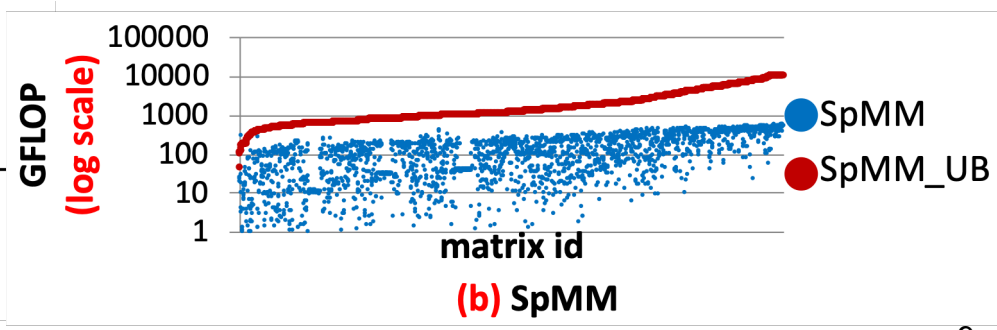
What about Sparse Tensor Computations?

- Many recent efforts on optimizing sparse tensor computations: TTM (Tensor Times Matrix), MTTKRP (Matricized Tensor Times Khatri-Rao Product)
  - Is data-movement a fundamental bottleneck? What are useful roofline limits?
- First seek insights from sparse matrix computation exemplars
  - Many key sparse tensor computations involve only one sparse tensor and one or more dense matrices
  - Use SpMM (Sparse-Dense Matrix-Matrix Multiplication) as sparse matrix exemplar to obtain insights

What is Achievable Performance Limit for SpMM?

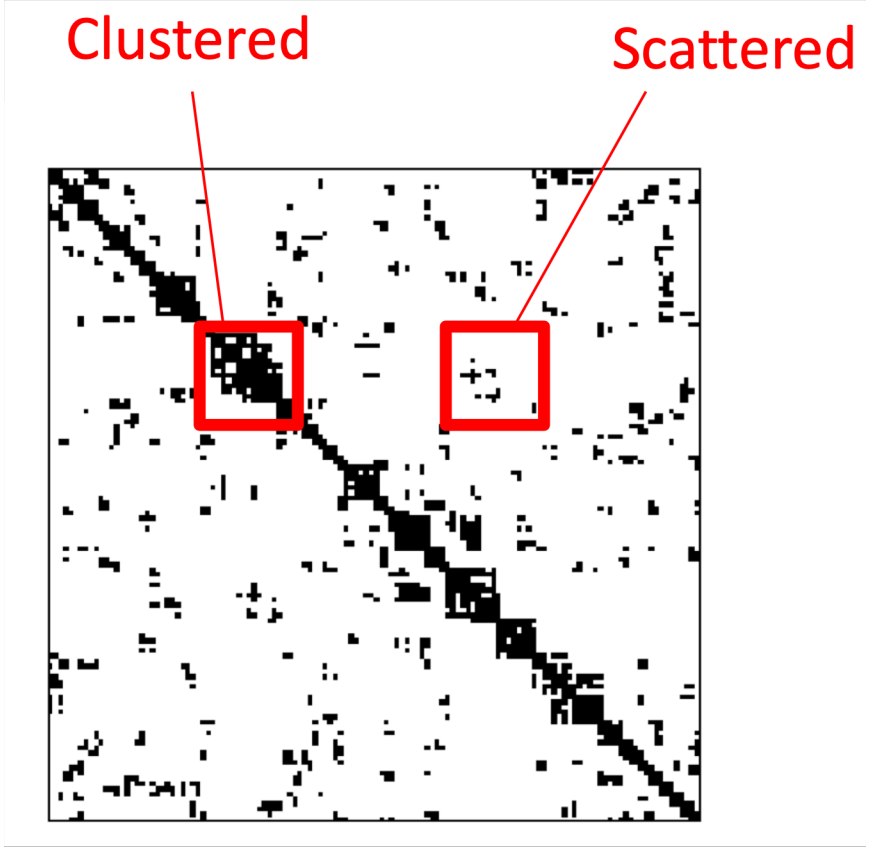
- Roofline Limits SpMV versus SpMM
  - SpMV: Each sparse matrix element is only used for one FMA; assuming at least two words per sparse-matrix element, upper-bound on OI is 1 FLOP/word = 0.25 Flops/byte (Single-Precision)
  - SpMM: (NxM Sparse) times (MxD Dense) => (NxD Dense)
  - Each sparse-matrix element is used for 2*D FLOPs; each element of dense input matrix is used for 2*nnz-avg$_{col}$ flops;
  - Often 1 < nnz-avg$_{col}$ < D, making the dense matrix data volume highest
  - Actually achieved performance (cuSPARSE) gets close to roofline limit for many matrices for SpMV but not for SpMM [SuiteSparse Data sets]





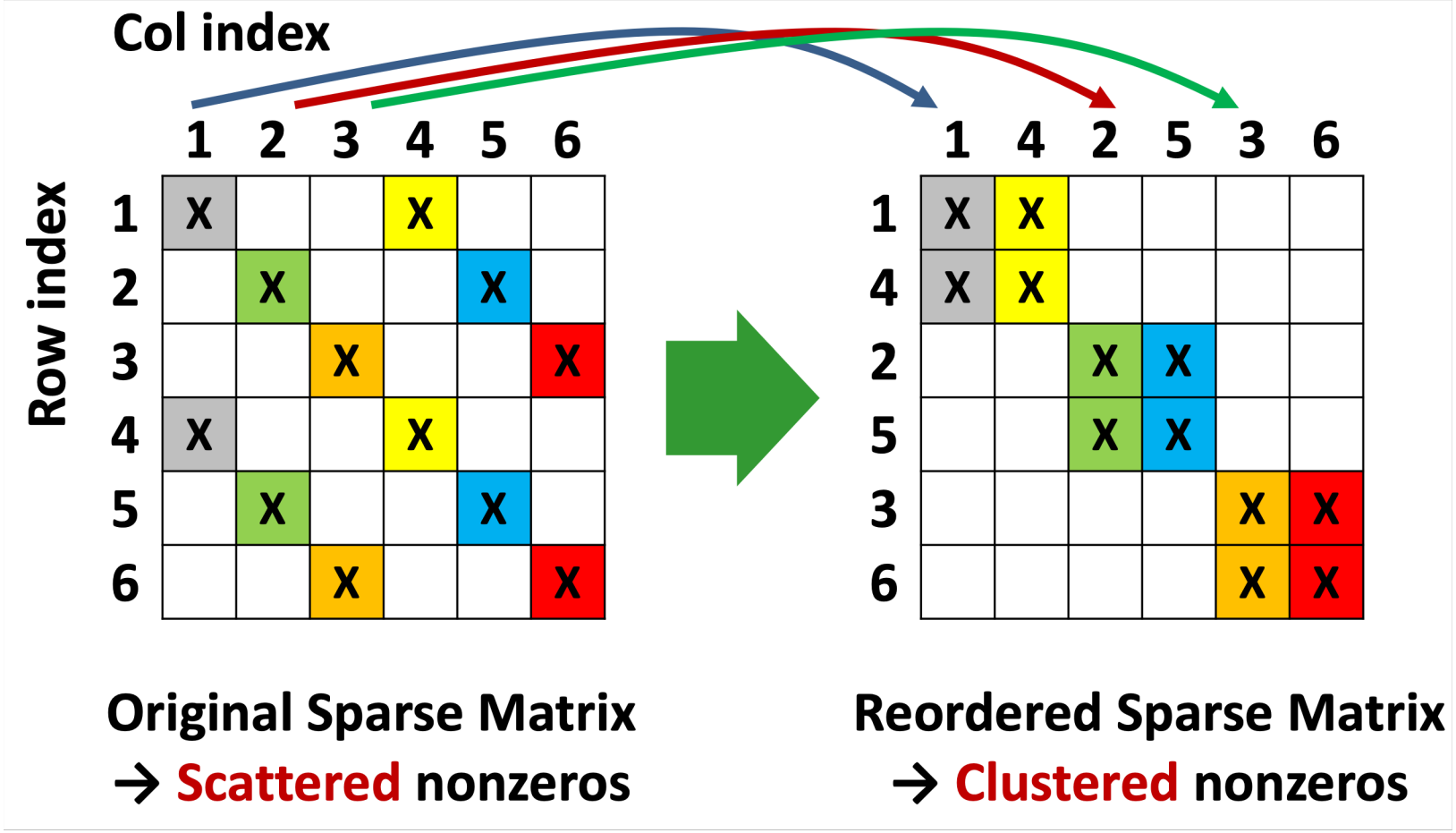
Non-Uniformity of Sparsity Patterns

- Performance of SpMM is very dependent on clustered versus scattered fraction of sparse-matrix non-zeros
- Work in progress: Develop useful abstraction of sparsity structure of a matrix to guide selection of effective tile size for SpMM



Can Matrix Reordering Improve Performance?

- Can clustering of non-zeros be improved via matrix reordering to improve SpMM performance?
- Work in progress: Evaluate existing matrix reordering schemes and explore new schemes



Summary

- Data locality optimization is increasingly important as machine balance parameters shift the critical Operational Intensity upwards
- Efficient implementation of dense tensor contraction on GPUs by adapting well-known scheme for dense matrix multiplication
- Sparse tensor computations are very challenging to optimize
  - Seek insights first from sparse matrix computations
- Open Question: Is Sparse-Dense Matrix Multiplication (SpMM) for representative matrices data-movement limited?
- Open Question: Can sparse matrix reordering have a significant impact on performance of SpMM?
- Same questions for sparse tensor computations more difficult to answer
- Key Need: A repository with a large number of realistic tensor datasets for various demanding tensor computations