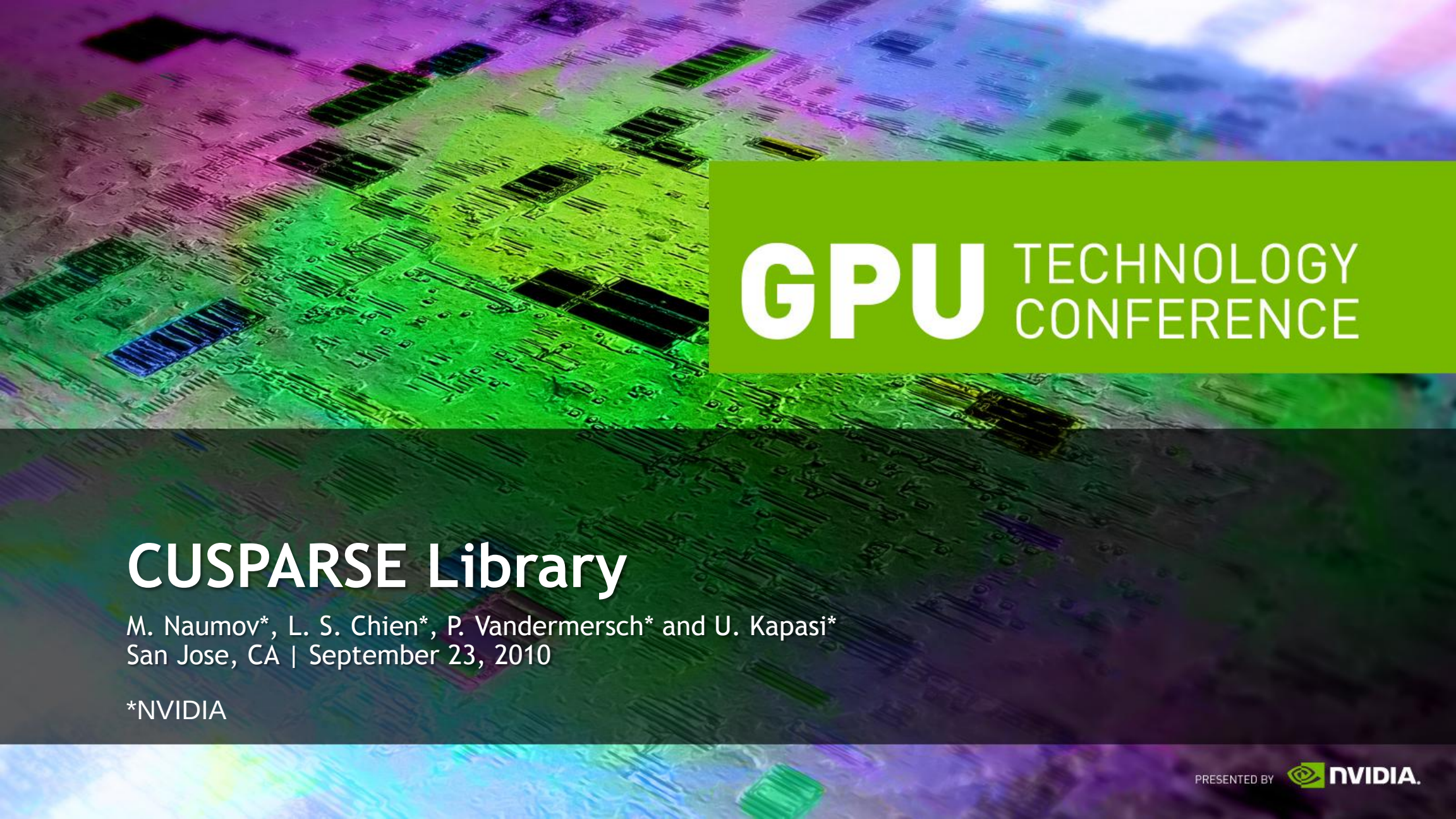


CUSPARSE Library

M. Naumov*, L. S. Chien*, P. Vandermersch* and U. Kapasi*

San Jose, CA | September 23, 2010

*NVIDIA

Outline

- Introduction to CUSPARSE library
  - ✓ Level-1, Level-2, Level-3 and Format Conversions
- Matrix-vector multiplication:
  - ✓ Detailed description, performance results
- Application Examples
  - ✓ Google PageRank Algorithm, Atmospheric Modeling
- Conclusion

Sparse Storage Formats

- Many choices for sparse matrix storage
  - ✓ COO (more general)
  - ✓ CSR
  - ✓ CSC
  - ✓ DIAG (more specific)
  - ✓ ELL
  - ✓ HYB (ELL + CSR)

- Things to keep in mind
  - ✓ focused on more than matrix-vector multiplication
  - ✓ focus on more general formats
  - ✓ might add new formats in the future

Sparse Storage Formats

- Coordinate (COO) Format

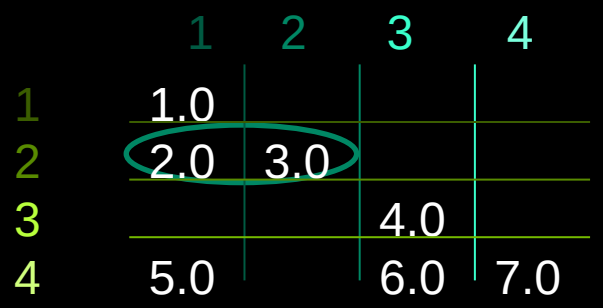
Row Index	1
Col Index	1
Values	1.0

	1	2	3	4
1	1.0			
2	2.0	3.0		
3			4.0	
4	5.0		6.0	7.0

Sparse Storage Formats

- Coordinate (COO) Format

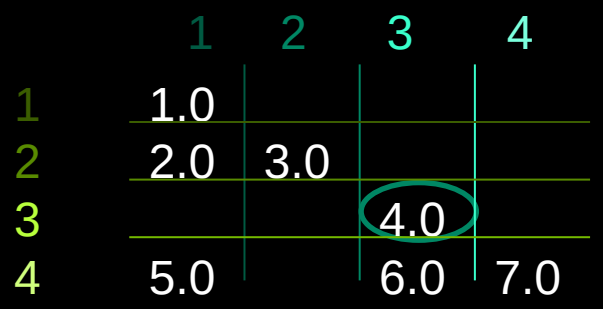
Row Index	1	2	2
Col Index	1	1	2
Values	1.0	2.0	3.0



Sparse Storage Formats

- Coordinate (COO) Format

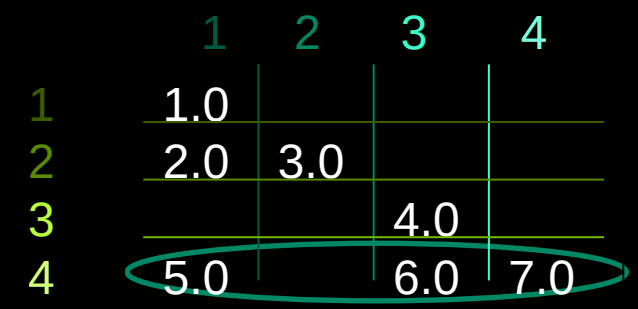
Row Index	1	2	2	3
Col Index	1	1	2	3
Values	1.0	2.0	3.0	4.0



Sparse Storage Formats

- Coordinate (COO) Format

Row Index	1	2	2	3	4	4	4
Col Index	1	1	2	3	1	3	4
Values	1.0	2.0	3.0	4.0	5.0	6.0	7.0



Sparse Storage Formats

Coordinate (COO) Format

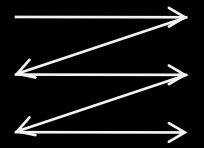
Row Index	1	2	2	3	4	4	4
Col Index	1	1	2	3	1	3	4
Values	1.0	2.0	3.0	4.0	5.0	6.0	7.0

	1	2	3	4
1	1.0			
2	2.0	3.0		
3			4.0	
4	5.0		6.0	7.0

Compressed Sparse Row (CSR)

Row Ptr	1						
Col Index	1	1	2	3	1	3	4
Values	1.0	2.0	3.0	4.0	5.0	6.0	7.0

row-major order



Sparse Storage Formats

Coordinate (COO) Format

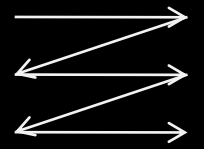
Row Index	1	2	2	3	4	4	4
Col Index	1	1	2	3	1	3	4
Values	1.0	2.0	3.0	4.0	5.0	6.0	7.0

	1	2	3	4
1	1.0			
2	2.0	3.0		
3			4.0	
4	5.0		6.0	7.0

Compressed Sparse Row (CSR)

Row Ptr	1	2		
Col Index	1	1	2	
Values	1.0	2.0	3.0	

row-major order



Sparse Storage Formats

➢ Coordinate (COO) Format

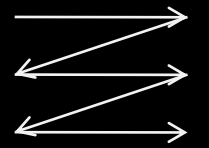
Row Index	1	2	2	3	4	4	4
Col Index	1	1	2	3	1	3	4
Values	1.0	2.0	3.0	4.0	5.0	6.0	7.0

	1	2	3	4
1	1.0			
2	2.0	3.0		
3			4.0	
4	5.0		6.0	7.0

➢ Compressed Sparse Row (CSR)

Row Ptr	1	2	4		
Col Index	1	1	2	3	
Values	1.0	2.0	3.0	4.0	

row-major order



Sparse Storage Formats

Coordinate (COO) Format

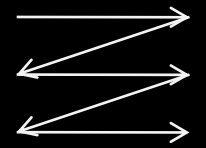
Row Index	1	2	2	3	4	4	4
Col Index	1	1	2	3	1	3	4
Values	1.0	2.0	3.0	4.0	5.0	6.0	7.0

	1	2	3	4
1	1.0			
2	2.0	3.0		
3			4.0	
4	5.0		6.0	7.0

Compressed Sparse Row (CSR)

Row Ptr	1	2	4	5	8		
Col Index	1	1	2	3	1	3	4
Values	1.0	2.0	3.0	4.0	5.0	6.0	7.0

row-major order



Sparse Storage Formats

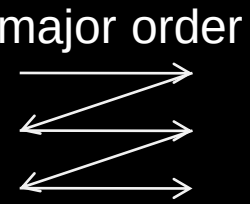
Coordinate (COO) Format

Row Index	1	2	2	3	4	4	4
Col Index	1	1	2	3	1	3	4
Values	1.0	2.0	3.0	4.0	5.0	6.0	7.0

	1	2	3	4
1	1.0			
2	2.0	3.0		
3			4.0	
4	5.0		6.0	7.0

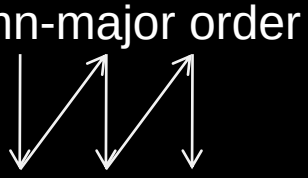
Compressed Sparse Row (CSR)

Row Ptr	1	2	4	5	8		
Col Index	1	1	2	3	1	3	4
Values	1.0	2.0	3.0	4.0	5.0	6.0	7.0



Compressed Sparse Column (CSC)

Row Index	1	2	4	2	3	4	4
Col Ptr	1	4	5	7	8		
Values	1.0	2.0	3.0	4.0	5.0	6.0	7.0



Basic Linear Algebra Subroutines

- Sparse vector storage

dense vector

1	2	3	4	5	6
1.0	0.0	0.0	2.0	0.0	3.0

Basic Linear Algebra Subroutines

- Sparse vector storage

sparse vector

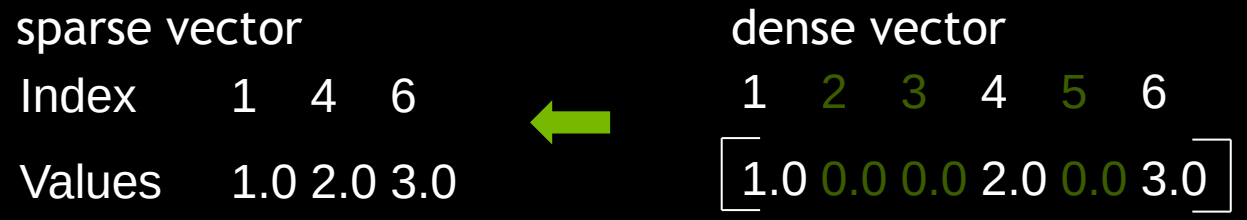
Index	1	4	6
Values	1.0	2.0	3.0

dense vector

1	2	3	4	5	6
1.0	0.0	0.0	2.0	0.0	3.0

Basic Linear Algebra Subroutines

- Sparse vector storage

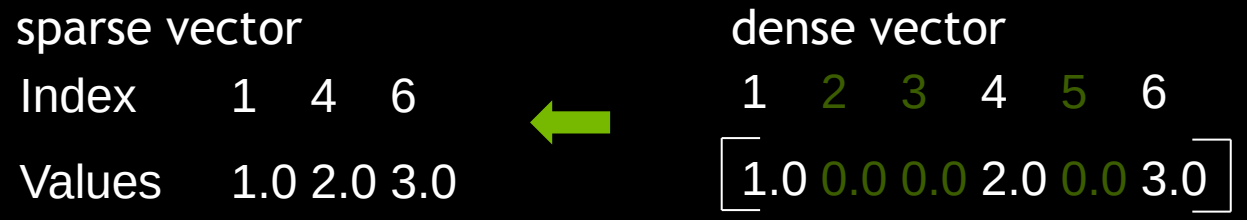


- Function naming convention
  - ✓ Follows the general rule

cusparse<Type>[<sparse data format>]<operation>[<sparse data format>]

Basic Linear Algebra Subroutines

- Sparse vector storage



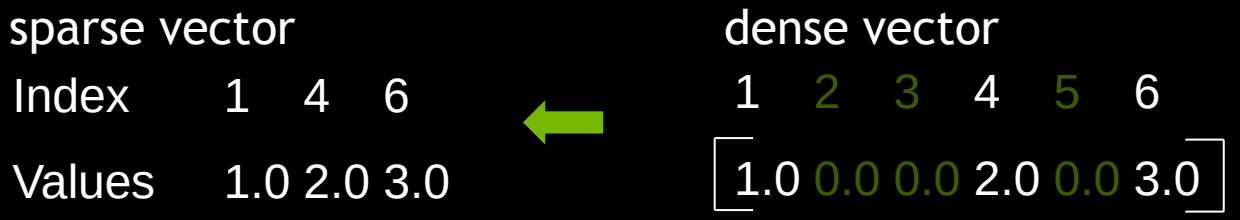
- Function naming convention
  - ✓ Follows the general rule

cusparse<Type>[<sparse data format>]<operation>[<sparse data format>]

  - ✓ For example,
    (i) single precision, sparse matrix (in csr storage) x dense vector =>
    cusparseScsrmv

Basic Linear Algebra Subroutines

- Sparse vector storage



- Function naming convention
  - ✓ Follows the general rule

cusparse<Type>[<sparse data format>]<operation>[<sparse data format>]

  - ✓ For example,
    (i) single precision, sparse matrix (in csr storage) x dense vector =>
    cusparseScsrmv
    (ii) double precision, sparse matrix (in csr storage) x dense tall-matrix => (set of vectors)
    cusparseDcsrmm

Basic Linear Algebra Subroutines

- Level 1 (sparse vector x and dense vector y)
  - ✓ axpyi (vector add) $y = y + \alpha x$

Basic Linear Algebra Subroutines

- Level 1 (sparse vector x and dense vector y)
  - ✓ axpyi (vector add) $y = y + \alpha x$
  - ✓ doti (dot product) $\alpha = y^T * x$

Basic Linear Algebra Subroutines

- Level 1 (sparse vector x and dense vector y)
  - ✓ axpyi (vector add) $y = y + \alpha x$
  - ✓ doti (dot product) $\alpha = y^T * x$
  - ✓ dotci (conjugate dot product) $\alpha = y^H * x$

Basic Linear Algebra Subroutines

- Level 1 (sparse vector x and dense vector y)
  - ✓ axpyi (vector add) $y = y + \alpha x$
  - ✓ doti (dot product) $\alpha = y^T * x$
  - ✓ dotci (conjugate dot product) $\alpha = y^H * x$
  - ✓ gthr (gather) $x = y$

Basic Linear Algebra Subroutines

- Level 1 (sparse vector x and dense vector y)
  - ✓ axpyi (vector add) — $y = y + \alpha x$
  - ✓ doti (dot product) — $\alpha = y^T * x$
  - ✓ dotci (conjugate dot product) — $\alpha = y^H * x$
  - ✓ gthr (gather) — $x = y$
  - ✓ gthrz (gather and zero-out) — $x = y$ and $x\_pattern(y)=0$

Basic Linear Algebra Subroutines

- Level 1 (sparse vector x and dense vector y)
  - ✓ axpyi (vector add) $y = y + \alpha x$
  - ✓ doti (dot product) $\alpha = y^T * x$
  - ✓ dotci (conjugate dot product) $\alpha = y^H * x$
  - ✓ gthr (gather) $x = y$
  - ✓ gthrz (gather and zero-out) $x = y$ and $x\_pattern(y)=0$
  - ✓ sctr (scatter) $y = x$

Basic Linear Algebra Subroutines

- Level 1 (sparse vector x and dense vector y)
  - ✓ axpyi (vector add) — $y = y + \alpha x$
  - ✓ doti (dot product) — $\alpha = y^T * x$
  - ✓ dotci (conjugate dot product) — $\alpha = y^H * x$
  - ✓ gthr (gather) — $x = y$
  - ✓ gthrz (gather and zero-out) — $x = y$ and $x\_pattern(y)=0$
  - ✓ sctr (scatter) — $y = x$
  - ✓ roti (Givens rotation) — $[x, y] = [x, y] \begin{bmatrix} c & -s \\ s & c \end{bmatrix}$

Basic Linear Algebra Subroutines

- Level 2 and 3
  - ✓ csrmv (matrix-vector multiplication)

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \alpha \begin{bmatrix} 1.0 & & & \\ 2.0 & 3.0 & & \\ & & 4.0 & \\ 5.0 & & 6.0 & 7.0 \end{bmatrix} \begin{bmatrix} 1.0 \\ 2.0 \\ 3.0 \\ 4.0 \end{bmatrix} + \beta \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

Basic Linear Algebra Subroutines

- Level 2 and 3
  - ✓ csrmv (matrix-vector multiplication)

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \alpha \begin{bmatrix} 1.0 & & & \\ 2.0 & 3.0 & & \\ & & 4.0 & \\ 5.0 & & 6.0 & 7.0 \end{bmatrix} \begin{bmatrix} 1.0 \\ 2.0 \\ 3.0 \\ 4.0 \end{bmatrix} + \beta \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

  - ✓ csrmm (matrix-tall-matrix multiplication)

$$\begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \\ x_{41} & x_{42} \end{bmatrix} = \alpha \begin{bmatrix} 1.0 & & & \\ 2.0 & 3.0 & & \\ & & 4.0 & \\ 5.0 & & 6.0 & 7.0 \end{bmatrix} \begin{bmatrix} 1.0 & 5.0 \\ 2.0 & 6.0 \\ 3.0 & 7.0 \\ 4.0 & 8.0 \end{bmatrix} + \beta \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \\ x_{41} & x_{42} \end{bmatrix}$$

Basic Linear Algebra Subroutines

- Level 2 and 3 (future work)
  - ✓ csrtrsv (triangular solve with a single right-hand-side)

$$\begin{bmatrix} 1.0 & & & \\ 2.0 & 3.0 & & \\ & & 4.0 & \\ 5.0 & & 6.0 & 7.0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \alpha \begin{bmatrix} 10.0 \\ 20.0 \\ 30.0 \\ 40.0 \end{bmatrix}$$

Basic Linear Algebra Subroutines

- Level 2 and 3 (future work)
  - ✓ csrtrsv (triangular solve with a single right-hand-side)

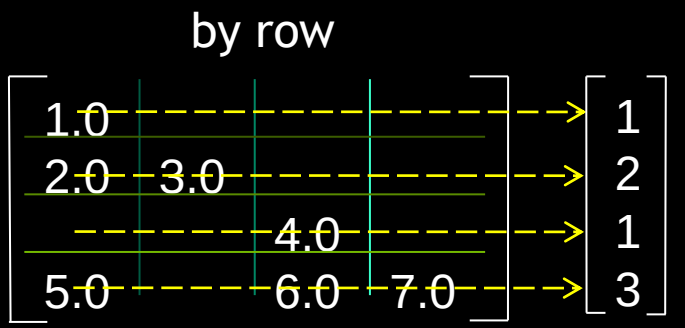
$$\begin{bmatrix} 1.0 & & & \\ 2.0 & 3.0 & & \\ & & 4.0 & \\ 5.0 & & 6.0 & 7.0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \alpha \begin{bmatrix} 10.0 \\ 20.0 \\ 30.0 \\ 40.0 \end{bmatrix}$$

  - ✓ csrtrsm (triangular solve with multiple right-hand-sides)

$$\begin{bmatrix} 1.0 & & & \\ 2.0 & 3.0 & & \\ & & 4.0 & \\ 5.0 & & 6.0 & 7.0 \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \\ x_{41} & x_{42} \end{bmatrix} = \alpha \begin{bmatrix} 10.0 & 50.0 \\ 20.0 & 60.0 \\ 30.0 & 70.0 \\ 40.0 & 80.0 \end{bmatrix}$$

Basic Linear Algebra Subroutines

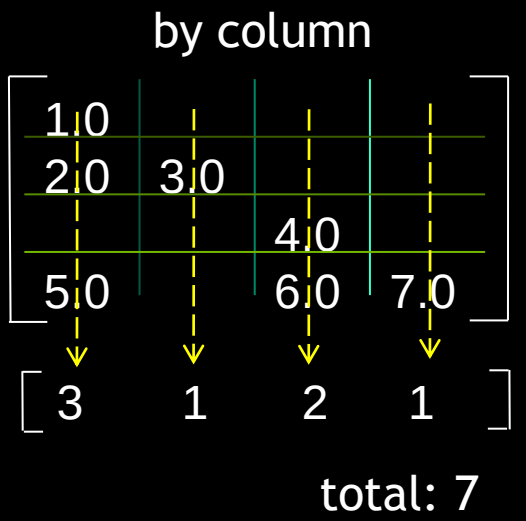
- Conversions
  - ✓ nnz
  - ✓ dense2csr
  - ✓ dense2csc
  - ✓ csr2dense
  - ✓ csc2dense
  - ✓ csr2coo
  - ✓ coo2csr
  - ✓ csr2csc



total: 7

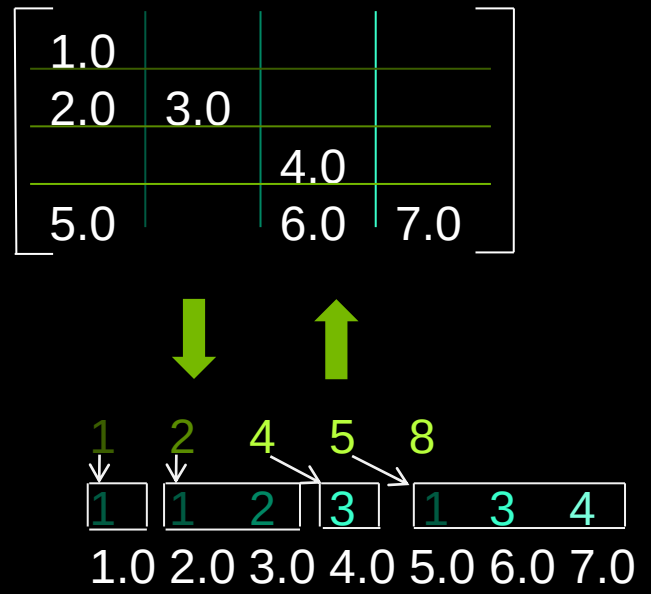
Basic Linear Algebra Subroutines

- Conversions
  - ✓ nnz
  - ✓ dense2csr
  - ✓ dense2csc
  - ✓ csr2dense
  - ✓ csc2dense
  - ✓ csr2coo
  - ✓ coo2csr
  - ✓ csr2csc



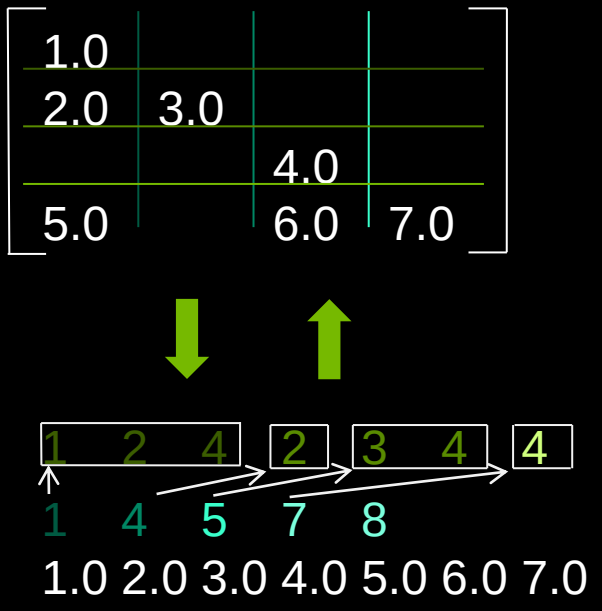
Basic Linear Algebra Subroutines

- Conversions
  - ✓ nnz
  - ✓ dense2csr
  - ✓ dense2csc
  - ✓ csr2dense
  - ✓ csc2dense
  - ✓ csr2coo
  - ✓ coo2csr
  - ✓ csr2csc



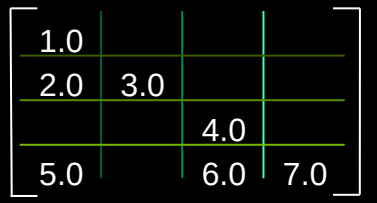
Basic Linear Algebra Subroutines

- Conversions
  - ✓ nnz
  - ✓ dense2csr
  - ✓ dense2csc
  - ✓ csr2dense
  - ✓ csc2dense
  - ✓ csr2coo
  - ✓ coo2csr
  - ✓ csr2csc

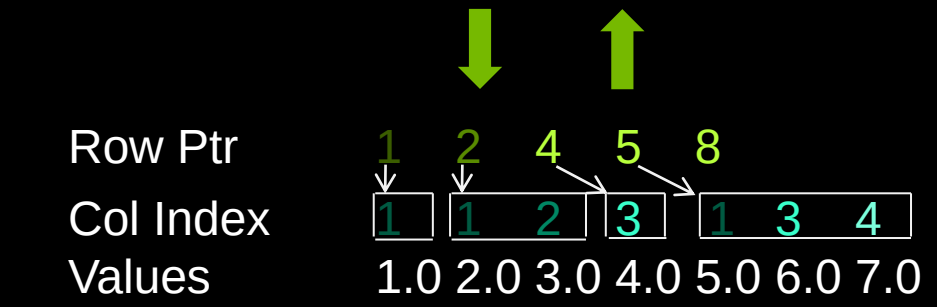


Basic Linear Algebra Subroutines

- Conversions
  - ✓ nnz
  - ✓ dense2csr
  - ✓ dense2csc
  - ✓ csr2dense
  - ✓ csc2dense
  - ✓ csr2coo
  - ✓ coo2csr
  - ✓ csr2csc

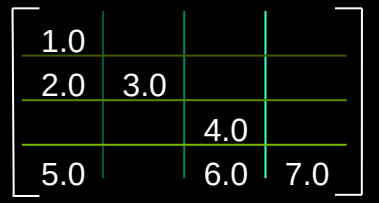


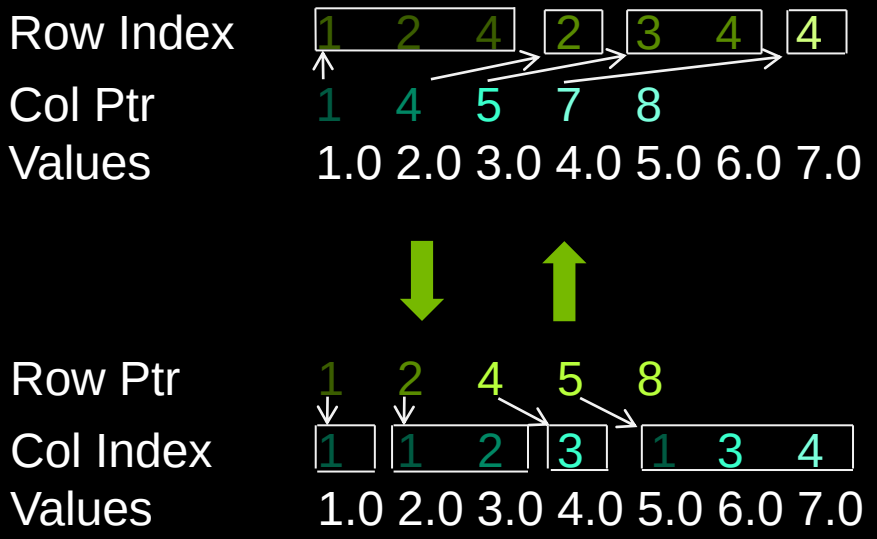
Row Index	1	2	2	3	4	4	4
Col Index	1	1	2	3	1	3	4
Values	1.0	2.0	3.0	4.0	5.0	6.0	7.0



Basic Linear Algebra Subroutines

- Conversions
  - ✓ nnz
  - ✓ dense2csr
  - ✓ dense2csc
  - ✓ csr2dense
  - ✓ csc2dense
  - ✓ csr2coo
  - ✓ coo2csr
  - ✓ csr2csc



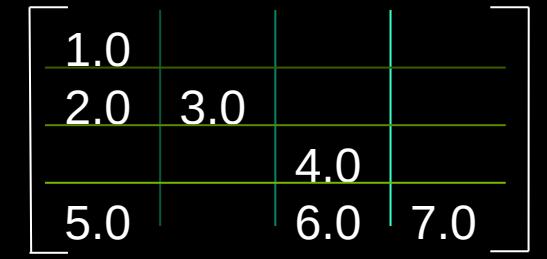


Detailed study of csrmv

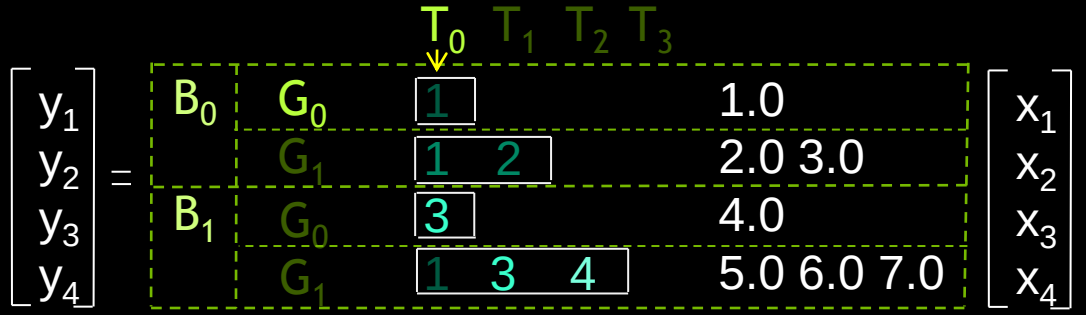
- Follow approach proposed by N. Bell & M. Garland (NVResearch)
  - ✓ Multiple blocks B_i, each processing a group G_j of rows
  - ✓ Each row is assigned a group of threads T_k
  - ✓ A few additional tweaks for performance

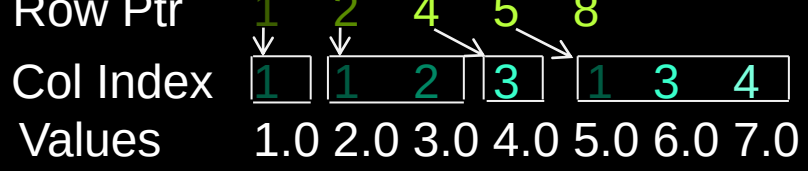
Detailed study of csrmv

- Follow approach proposed by N. Bell & M. Garland (NVResearch)
  - ✓ Multiple blocks B_i, each processing a group G_j of rows
  - ✓ Each row is assigned a group of threads T_k
  - ✓ A few additional tweaks for performance



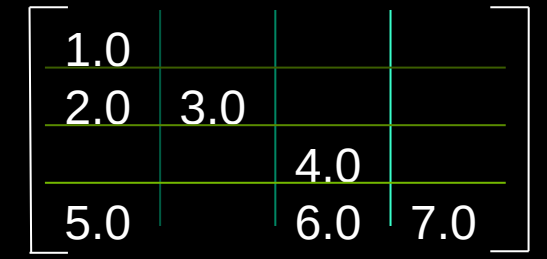
✓ Consider 2 blocks, with 2 groups of 4 threads; matrix:



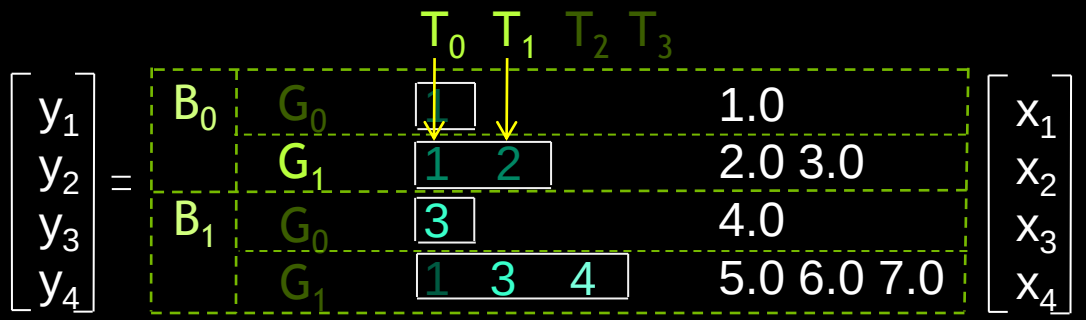


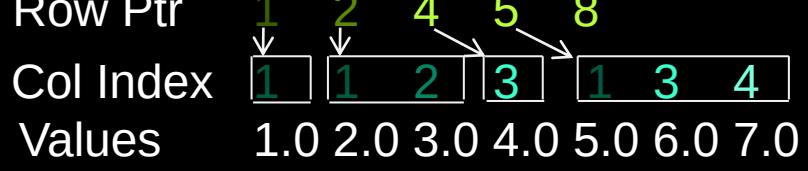
Detailed study of csrmv

- Follow approach proposed by N. Bell & M. Garland (NVResearch)
  - ✓ Multiple blocks B_i, each processing a group G_j of rows
  - ✓ Each row is assigned a group of threads T_k
  - ✓ A few additional tweaks for performance



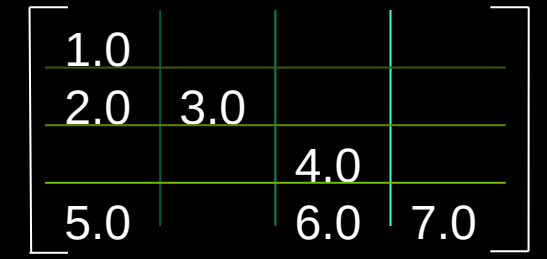
✓ Consider 2 blocks, with 2 groups of 4 threads; matrix:



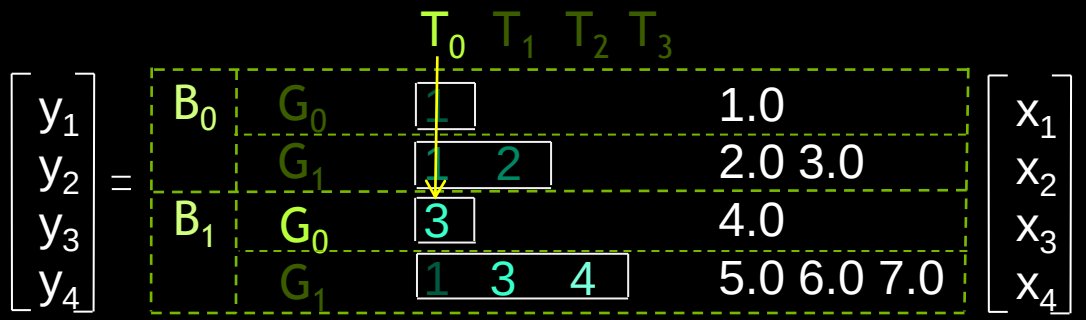


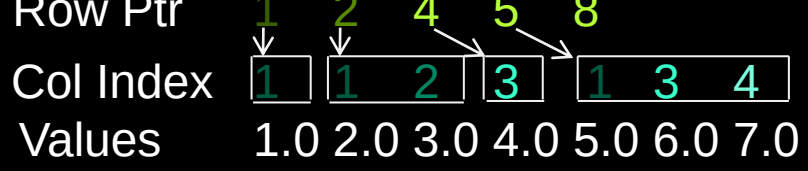
Detailed study of csrmv

- Follow approach proposed by N. Bell & M. Garland (NVResearch)
  - ✓ Multiple blocks B_i, each processing a group G_j of rows
  - ✓ Each row is assigned a group of threads T_k
  - ✓ A few additional tweaks for performance



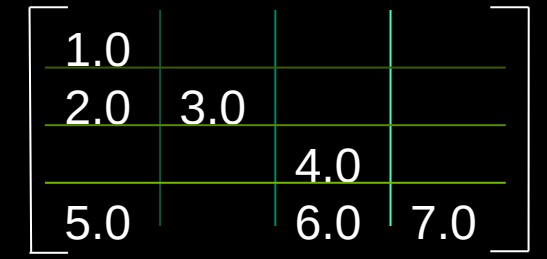
✓ Consider 2 blocks, with 2 groups of 4 threads; matrix:



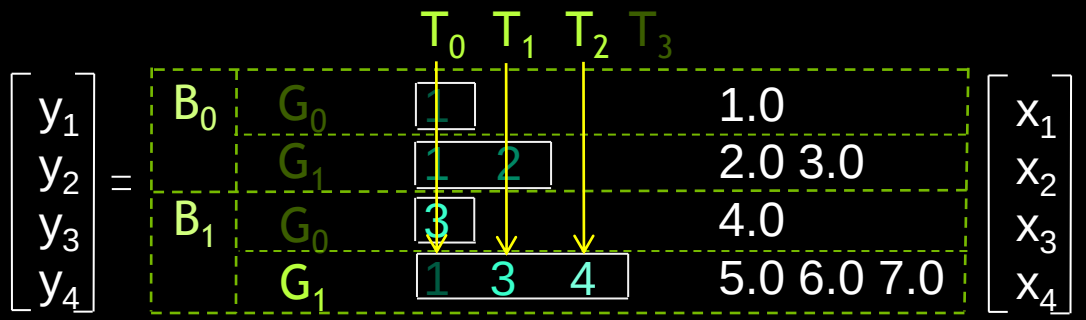


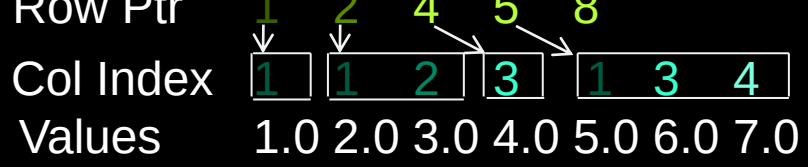
Detailed study of csrmv

- Follow approach proposed by N. Bell & M. Garland (NVResearch)
  - ✓ Multiple blocks B_i, each processing a group G_j of rows
  - ✓ Each row is assigned a group of threads T_k
  - ✓ A few additional tweaks for performance



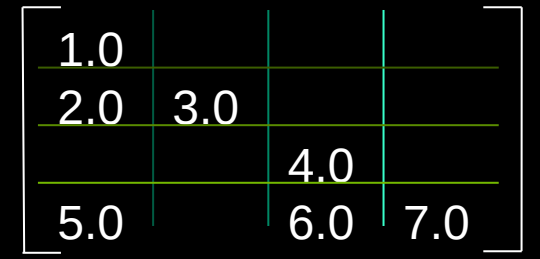
✓ Consider 2 blocks, with 2 groups of 4 threads; matrix:



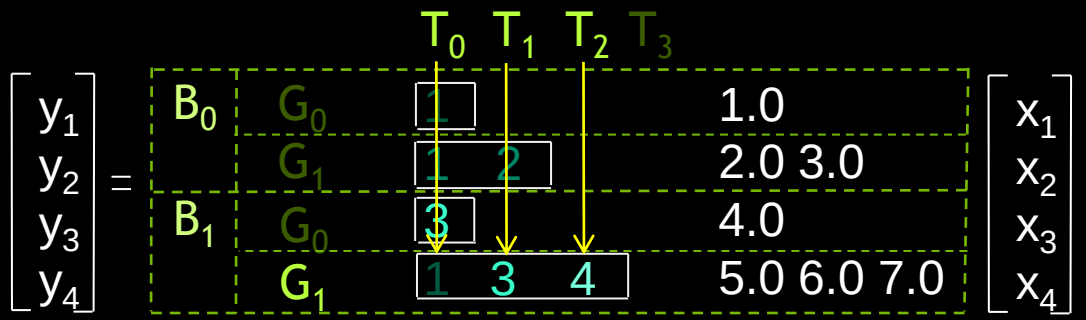


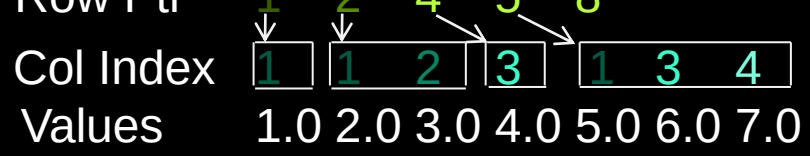
Detailed study of csrmv

- Follow approach proposed by N. Bell & M. Garland (NVResearch)
  - ✓ Multiple blocks B_i, each processing a group G_j of rows
  - ✓ Each row is assigned a group of threads T_k
  - ✓ A few additional tweaks for performance
    - i. adjust number of threads per row to minimize waste



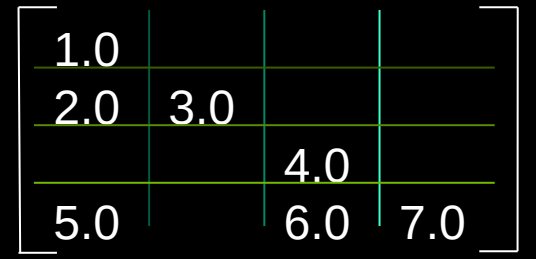
- ✓ Consider 2 blocks, with 2 groups of 4 threads; matrix:



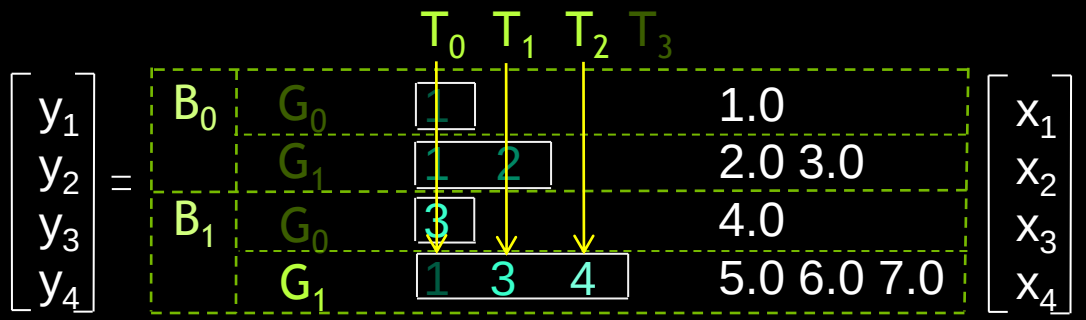


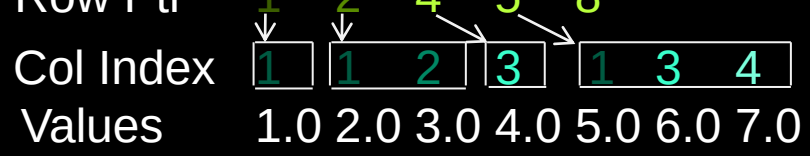
Detailed study of csrmv

- Follow approach proposed by N. Bell & M. Garland (NVResearch)
  - ✓ Multiple blocks B_i, each processing a group G_j of rows
  - ✓ Each row is assigned a group of threads T_k
  - ✓ A few additional tweaks for performance
    - i. adjust number of threads per row to minimize waste
    - ii. align threads per row for coalescing



- ✓ Consider 2 blocks, with 2 groups of 4 threads; matrix:

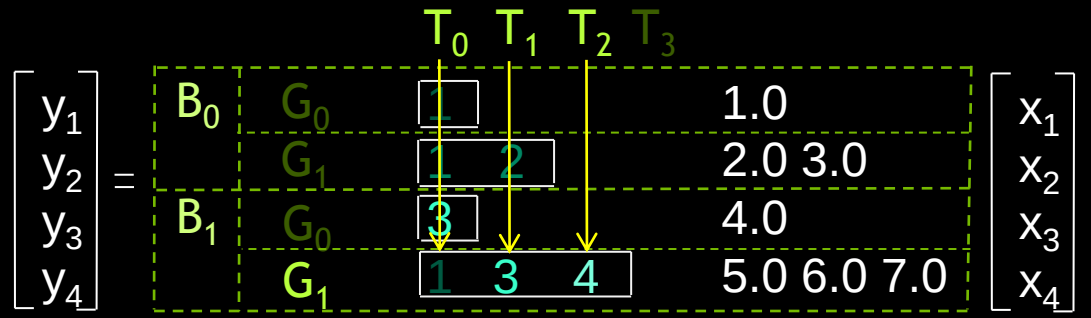


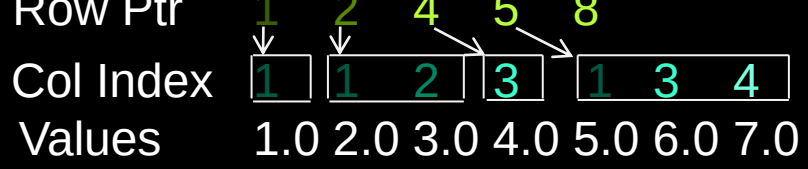


Detailed study of csrmv

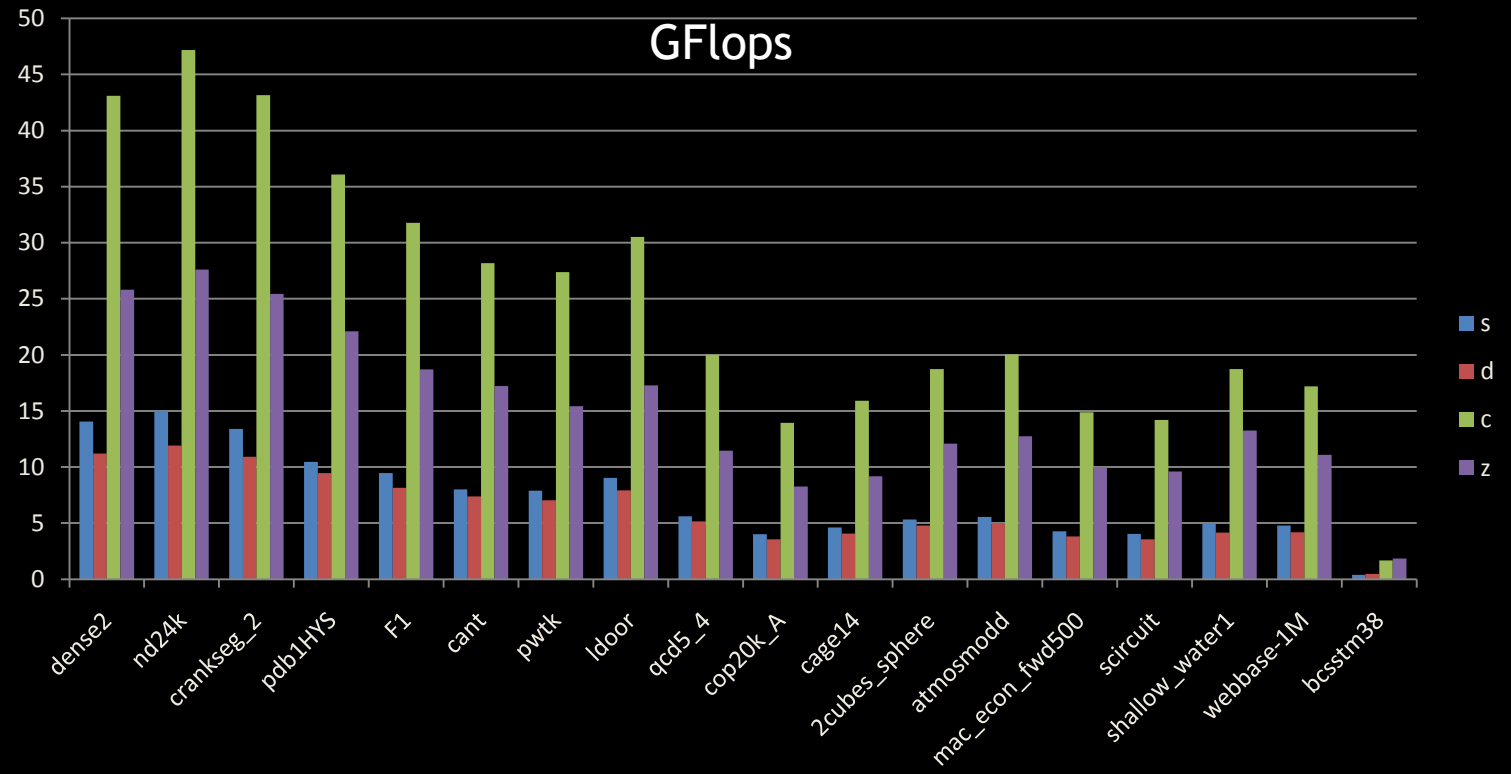
- Follow approach proposed by N. Bell & M. Garland (NVResearch)
  - ✓ Multiple blocks B_i, each processing a group G_j of rows
  - ✓ Each row is assigned a group of threads T_k
  - ✓ A few additional tweaks for performance
    - i. adjust number of threads per row to minimize waste
    - ii. align threads per row for coalescing
    - iii. use shared memory and texture for performance
  - ✓ Consider 2 blocks, with 2 groups of 4 threads; matrix:

$$\begin{bmatrix} 1.0 & & & \\ 2.0 & 3.0 & & \\ & & 4.0 & \\ 5.0 & & 6.0 & 7.0 \end{bmatrix}$$



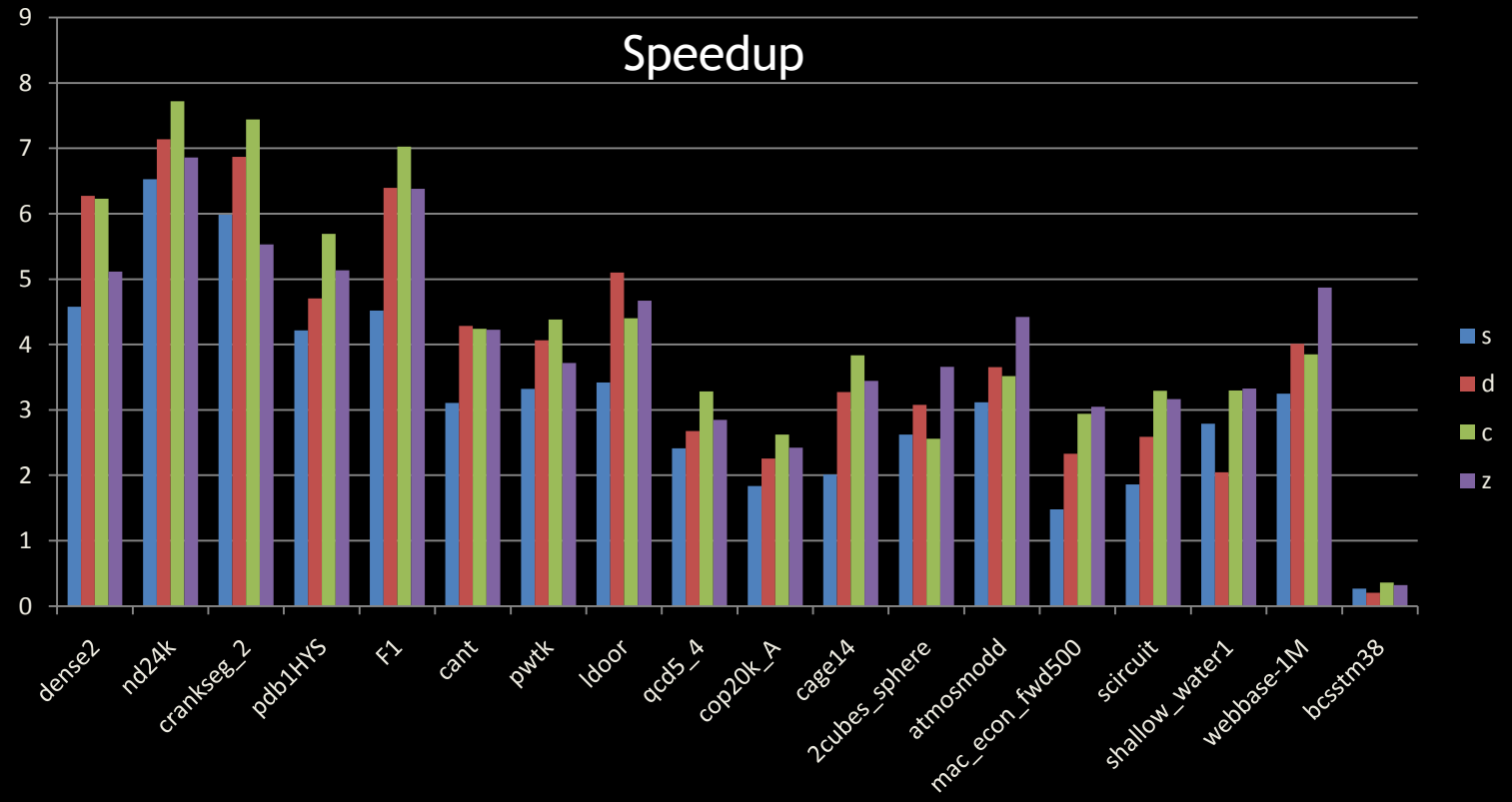


Performance of csrmv



*NVIDIA C2050, ECC on

Performance of csrmv



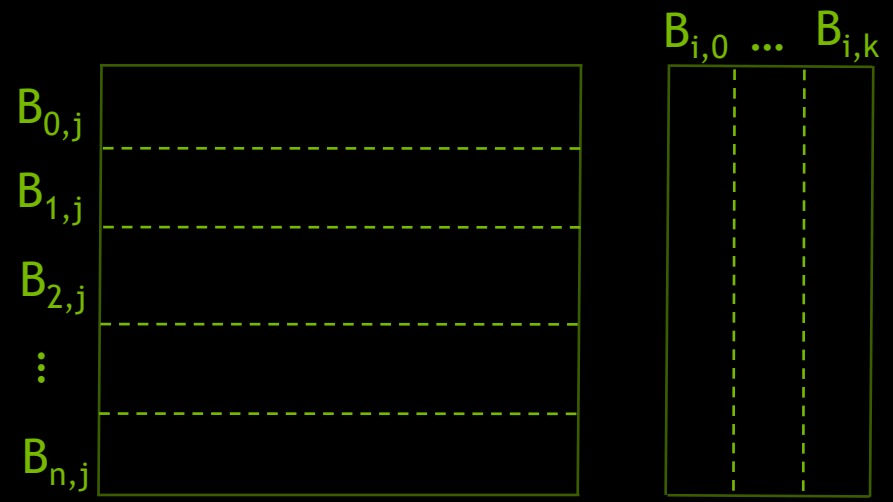
*NVIDIA C2050, ECC on
*MKL 10.2.3 , Core™ i7 @ 3.07GHz

Detailed study of csrmm

- How about multiplication by multiple vectors (tall-matrices)
  - ✓ Use similar approach for processing the sparse matrix

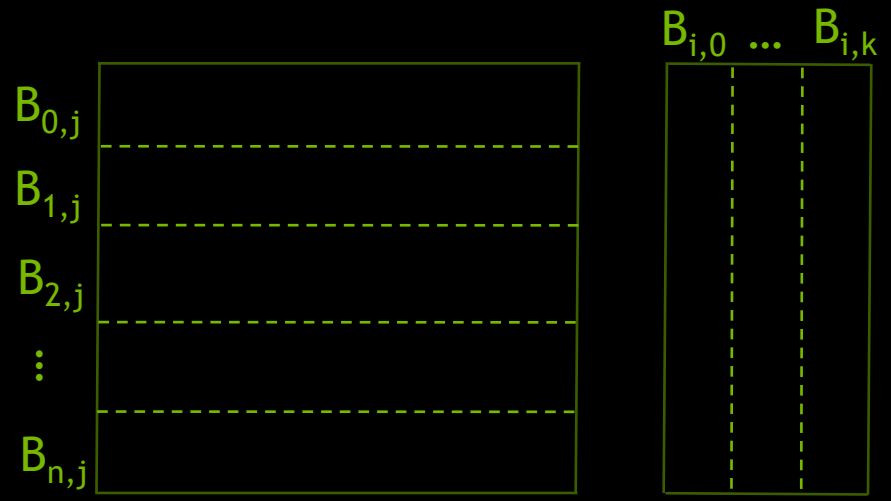
Detailed study of csrmm

- How about multiplication by multiple vectors (tall-matrices)
  - ✓ Use similar approach for processing the sparse matrix
  - ✓ Have a 2D grid of blocks (with second dimension processing columns of the tall-matrix)



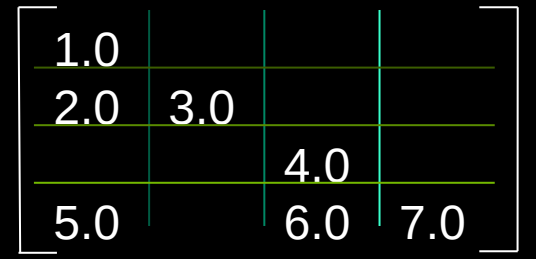
Detailed study of csrmm

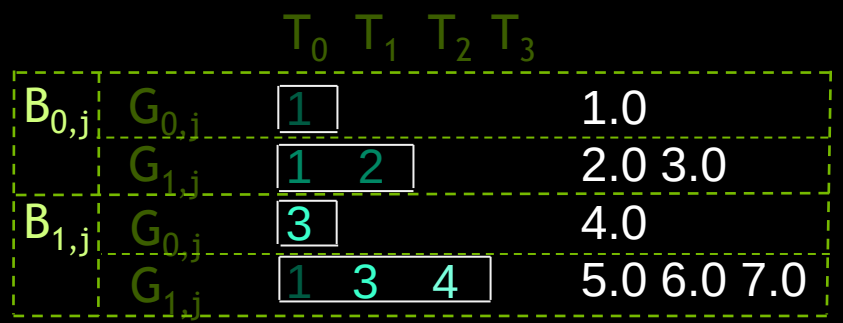
- How about multiplication by multiple vectors (tall-matrices)
  - ✓ Use similar approach for processing the sparse matrix
  - ✓ Have a 2D grid of blocks (with second dimension processing columns of the tall-matrix)
  - ✓ Each block processes multiple columns (to minimize reading sparse matrix multiple times)

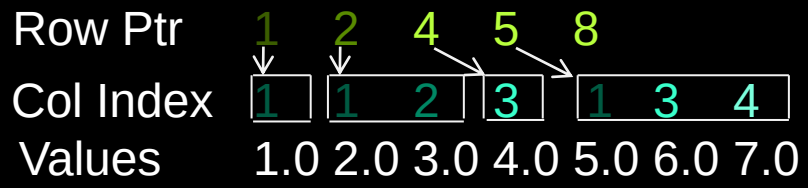


Detailed study of csrmm

- How about multiplication by multiple vectors (tall-matrices)
  - ✓ Use similar approach for processing the sparse matrix
  - ✓ Have a 2D grid of blocks (with second dimension processing columns of the tall-matrix)
  - ✓ Each block processes multiple columns (to minimize reading sparse matrix multiple times)
  - ✓ Consider 2x2 grid of blocks, with 2 groups of 4 threads; matrix:

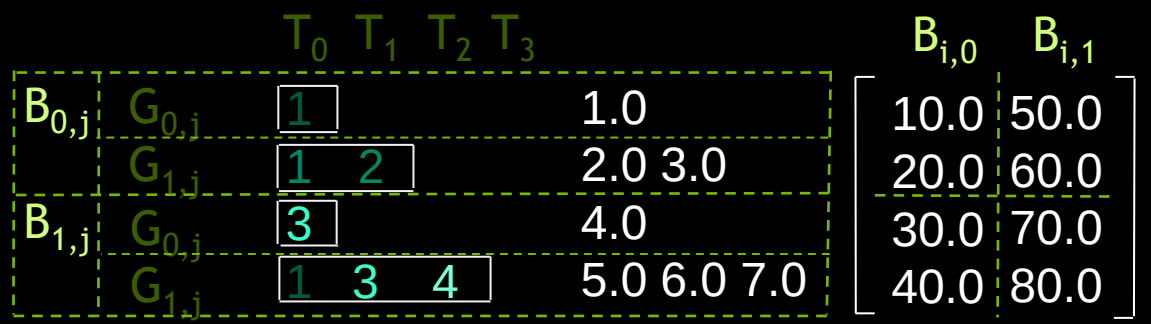


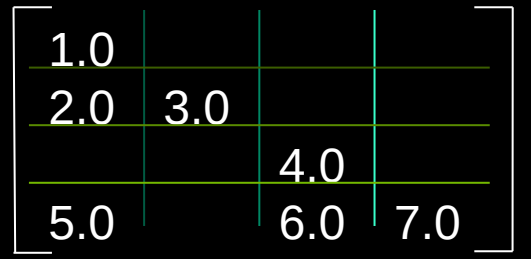


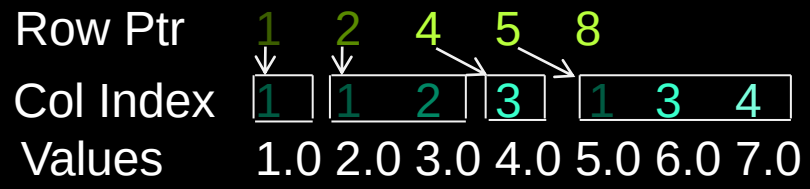


Detailed study of csrmm

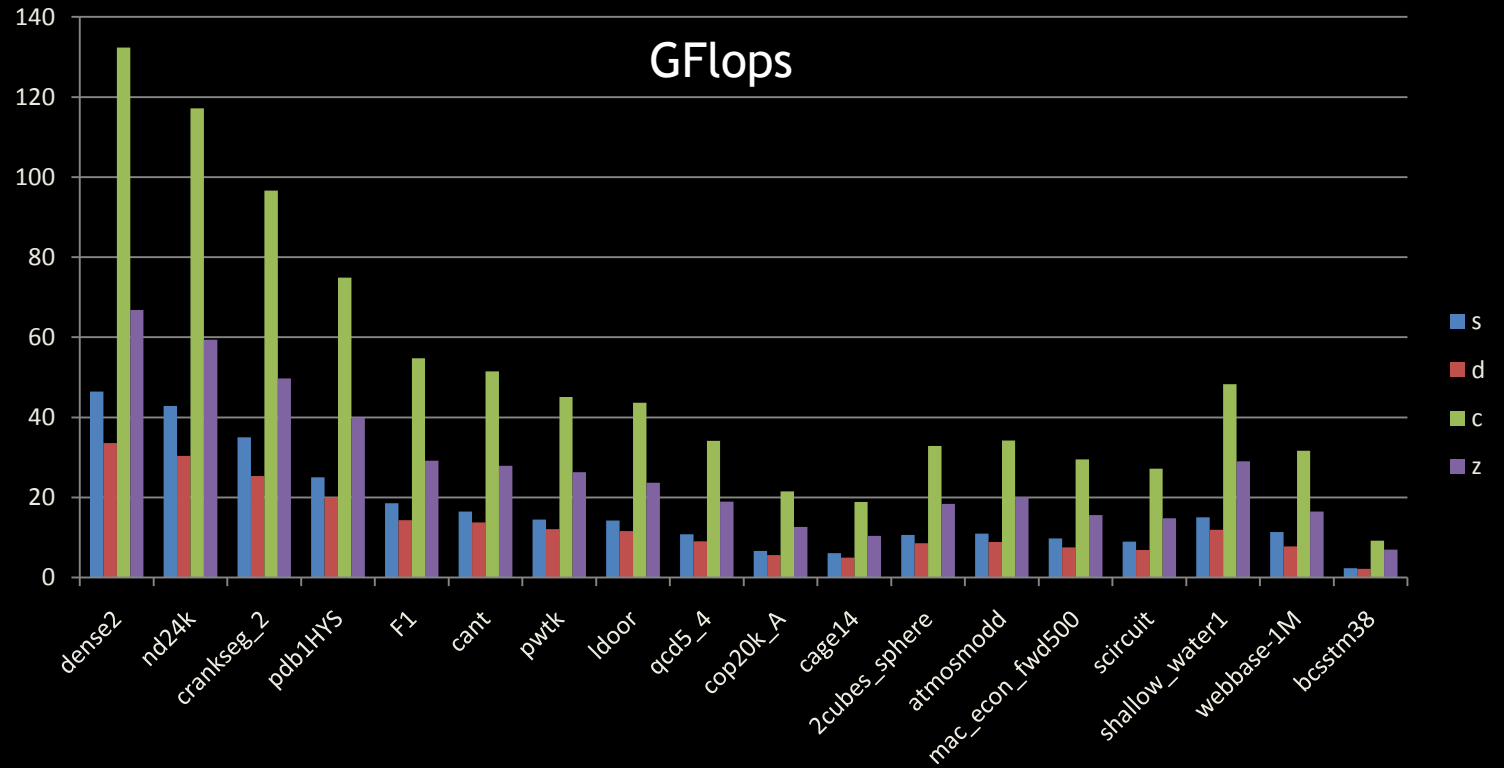
- How about multiplication by multiple vectors (tall-matrices)
  - ✓ Use similar approach for processing the sparse matrix
  - ✓ Have a 2D grid of blocks (with second dimension processing columns of the tall-matrix)
  - ✓ Each block processes multiple columns (to minimize reading sparse matrix multiple times)
  - ✓ Consider 2x2 grid of blocks, with 2 groups of 4 threads; matrix:





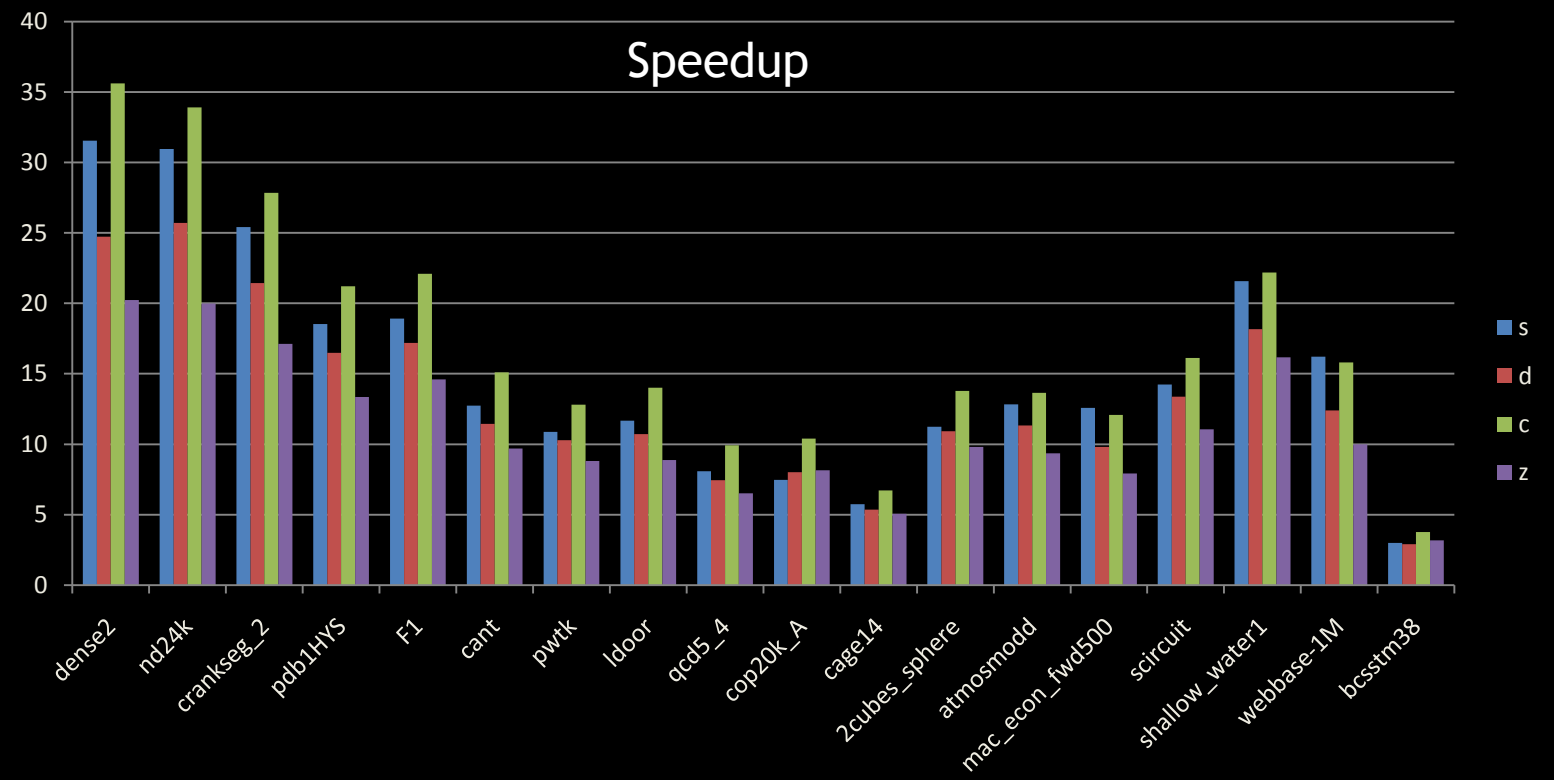


Performance of csrmm



*NVIDIA C2050, ECC on

Performance of csrmm



*NVIDIA C2050, ECC on
*MKL 10.2.3 , Core™ i7 @ 3.07GHz

Detailed study of csrmv transpose

- What is the difficulty?
  - ✓ Multiple blocks/threads writing to the same memory location

Detailed study of csrmv transpose

- What is the difficulty?
  - ✓ Multiple blocks/threads writing to the same memory location
  - ✓ Must use atomics to ensure correct results

Detailed study of csrmv transpose

- What is the difficulty?
  - ✓ Multiple blocks/threads writing to the same memory location
  - ✓ Must use atomics to ensure correct results

matrix storage view:

$$A = \begin{bmatrix} a_1^T \\ a_2^T \\ a_3^T \\ a_4^T \end{bmatrix} = \begin{bmatrix} 1.0 & & & \\ 2.0 & 3.0 & & \\ & & 4.0 & \\ 5.0 & & 6.0 & 7.0 \end{bmatrix}$$

Detailed study of csrmv transpose

- What is the difficulty?
  - ✓ Multiple blocks/threads writing to the same memory location
  - ✓ Must use atomics to ensure correct results

matrix (transpose) logical view:

$$A^T = \begin{bmatrix} a_1 & a_2 & a_3 & a_4 \end{bmatrix} = \begin{bmatrix} 1.0 & 2.0 & & 5.0 \\ & 3.0 & & \\ & & 4.0 & 6.0 \\ & & & 7.0 \end{bmatrix}$$

matrix storage view:

$$A = \begin{bmatrix} a_1^T \\ a_2^T \\ a_3^T \\ a_4^T \end{bmatrix} = \begin{bmatrix} 1.0 & & & \\ 2.0 & 3.0 & & \\ & & 4.0 & \\ 5.0 & & 6.0 & 7.0 \end{bmatrix}$$

Detailed study of csrmv transpose

- What is the difficulty?
  - ✓ Multiple blocks/threads writing to the same memory location
  - ✓ Must use atomics to ensure correct results

matrix (transpose) logical view:

$$A^T = \begin{bmatrix} a_1 & a_2 & a_3 & a_4 \end{bmatrix} = \left[\begin{array}{cccc} 1.0 & 2.0 & & 5.0 \\ & 3.0 & & \\ & & 4.0 & 6.0 \\ & & & 7.0 \end{array}\right]$$

G_0 G_1 G_0 G_1

B_0 B_1

matrix storage view:

$$A = \begin{bmatrix} a_1^T \\ a_2^T \\ a_3^T \\ a_4^T \end{bmatrix} = \left[\begin{array}{cccc} 1.0 & & & \\ 2.0 & 3.0 & & \\ & & 4.0 & \\ 5.0 & & 6.0 & 7.0 \end{array}\right]$$

Detailed study of csrmv transpose

- What is the difficulty?
  - ✓ Multiple blocks/threads writing to the same memory location
  - ✓ Must use atomics to ensure correct results

matrix (transpose) logical view:

$$A^T = [a_1\ a_2\ a_3\ a_4] = \begin{bmatrix} 1.0 & 2.0 & & 5.0 \\ & 3.0 & & \\ & & 4.0 & 6.0 \\ & & & 7.0 \end{bmatrix}$$

$G_0\ G_1\ G_0\ G_1$

$B_0 \qquad B_1$

matrix storage view:

$$A = \begin{bmatrix} a_1^T \\ a_2^T \\ a_3^T \\ a_4^T \end{bmatrix} = \begin{bmatrix} 1.0 & & & \\ 2.0 & 3.0 & & \\ & & 4.0 & \\ 5.0 & & 6.0 & 7.0 \end{bmatrix}$$

$$y = [a_1\ a_2\ a_3\ a_4] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

$$y=a_1*x_1+a_2*x_2+a_3*x_3+a_4*x_4$$

Detailed study of csrmv transpose

- What is the difficulty?
  - ✓ Multiple blocks/threads writing to the same memory location
  - ✓ Must use atomics to ensure correct results

matrix (transpose) logical view:

$$A^T = \begin{bmatrix} a_1 & a_2 & a_3 & a_4 \end{bmatrix} = \begin{bmatrix} 1.0 & 2.0 & & 5.0 \\ & 3.0 & & \\ & & 4.0 & 6.0 \\ & & & 7.0 \end{bmatrix}$$

G_0 G_1 G_0 G_1

B_0 B_1

matrix storage view:

$$A = \begin{bmatrix} a_1^T \\ a_2^T \\ a_3^T \\ a_4^T \end{bmatrix} = \begin{bmatrix} 1.0 & & & \\ 2.0 & 3.0 & & \\ & & 4.0 & \\ 5.0 & & 6.0 & 7.0 \end{bmatrix}$$

$$y = \begin{bmatrix} a_1 & a_2 & a_3 & a_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

$$\begin{bmatrix} 1.0 & 2.0 & & 5.0 \\ & 3.0 & & \\ & & 4.0 & 6.0 \\ & & & 7.0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

$y=a_1*x_1+a_2*x_2+a_3*x_3+a_4*x_4$

Detailed study of csrmv transpose

- What is the difficulty?
  - ✓ Multiple blocks/threads writing to the same memory location
  - ✓ Must use atomics to ensure correct results

matrix (transpose) logical view:

$$A^T = [a_1\ a_2\ a_3\ a_4] = \begin{bmatrix} 1.0 & 2.0 & & 5.0 \\ & 3.0 & & \\ & & 4.0 & 6.0 \\ & & & 7.0 \end{bmatrix}$$

$G_0\ G_1\ G_0\ G_1$

$B_0 \qquad B_1$

matrix storage view:

$$A = \begin{bmatrix} a_1^T \\ a_2^T \\ a_3^T \\ a_4^T \end{bmatrix} = \begin{bmatrix} 1.0 & & & \\ 2.0 & 3.0 & & \\ & & 4.0 & \\ 5.0 & & 6.0 & 7.0 \end{bmatrix}$$

$$y = [a_1\ a_2\ a_3\ a_4] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

$$\begin{bmatrix} 1.0 & 2.0 & & 5.0 \\ & 3.0 & & \\ & & 4.0 & 6.0 \\ & & & 7.0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

$$y=a_1*x_1+a_2*x_2+a_3*x_3+a_4*x_4$$

Detailed study of csrmv transpose

- What is the difficulty?
  - ✓ Multiple blocks/threads writing to the same memory location
  - ✓ Must use atomics to ensure correct results

matrix (transpose) logical view:

$$A^T = \begin{bmatrix} a_1 & a_2 & a_3 & a_4 \end{bmatrix} = \begin{bmatrix} 1.0 & 2.0 & & 5.0 \\ & 3.0 & & \\ & & 4.0 & 6.0 \\ & & & 7.0 \end{bmatrix}$$

G_0 G_1 G_0 G_1

B_0 B_1

matrix storage view:

$$A = \begin{bmatrix} a_1^T \\ a_2^T \\ a_3^T \\ a_4^T \end{bmatrix} = \begin{bmatrix} 1.0 & & & \\ 2.0 & 3.0 & & \\ & & 4.0 & \\ 5.0 & & 6.0 & 7.0 \end{bmatrix}$$

$$y = \begin{bmatrix} a_1 & a_2 & a_3 & a_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

$$\begin{bmatrix} 1.0 & 2.0 & & 5.0 \\ & 3.0 & & \\ & & 4.0 & 6.0 \\ & & & 7.0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

$y=a_1*x_1+a_2*x_2+a_3*x_3+a_4*x_4$

Detailed study of csrmv transpose

- What is the difficulty?
  - ✓ Multiple blocks/threads writing to the same memory location
  - ✓ Must use atomics to ensure correct results

matrix (transpose) logical view:

$$A^T = \begin{bmatrix} a_1 & a_2 & a_3 & a_4 \end{bmatrix} = \begin{bmatrix} 1.0 & 2.0 & & 5.0 \\ & 3.0 & & \\ & & 4.0 & 6.0 \\ & & & 7.0 \end{bmatrix}$$

G_0 G_1 G_0 G_1

B_0 B_1

matrix storage view:

$$A = \begin{bmatrix} a_1^T \\ a_2^T \\ a_3^T \\ a_4^T \end{bmatrix} = \begin{bmatrix} 1.0 & & & \\ 2.0 & 3.0 & & \\ & & 4.0 & \\ 5.0 & & 6.0 & 7.0 \end{bmatrix}$$

$$y = \begin{bmatrix} a_1 & a_2 & a_3 & a_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

$$\begin{bmatrix} 1.0 & 2.0 & & 5.0 \\ & 3.0 & & \\ & & 4.0 & 6.0 \\ & & & 7.0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

$$y=a_1*x_1+a_2*x_2+a_3*x_3+a_4*x_4$$

Detailed study of csrmv transpose

- What is the difficulty?
  - ✓ Multiple blocks/threads writing to the same memory location
  - ✓ Must use atomics to ensure correct results

matrix (transpose) logical view:

$$A^T = \begin{bmatrix} a_1 & a_2 & a_3 & a_4 \end{bmatrix} = \begin{bmatrix} 1.0 & 2.0 & & 5.0 \\ & 3.0 & & \\ & & 4.0 & 6.0 \\ & & & 7.0 \end{bmatrix}$$

G_0 G_1 G_0 G_1

B_0 B_1

matrix storage view:

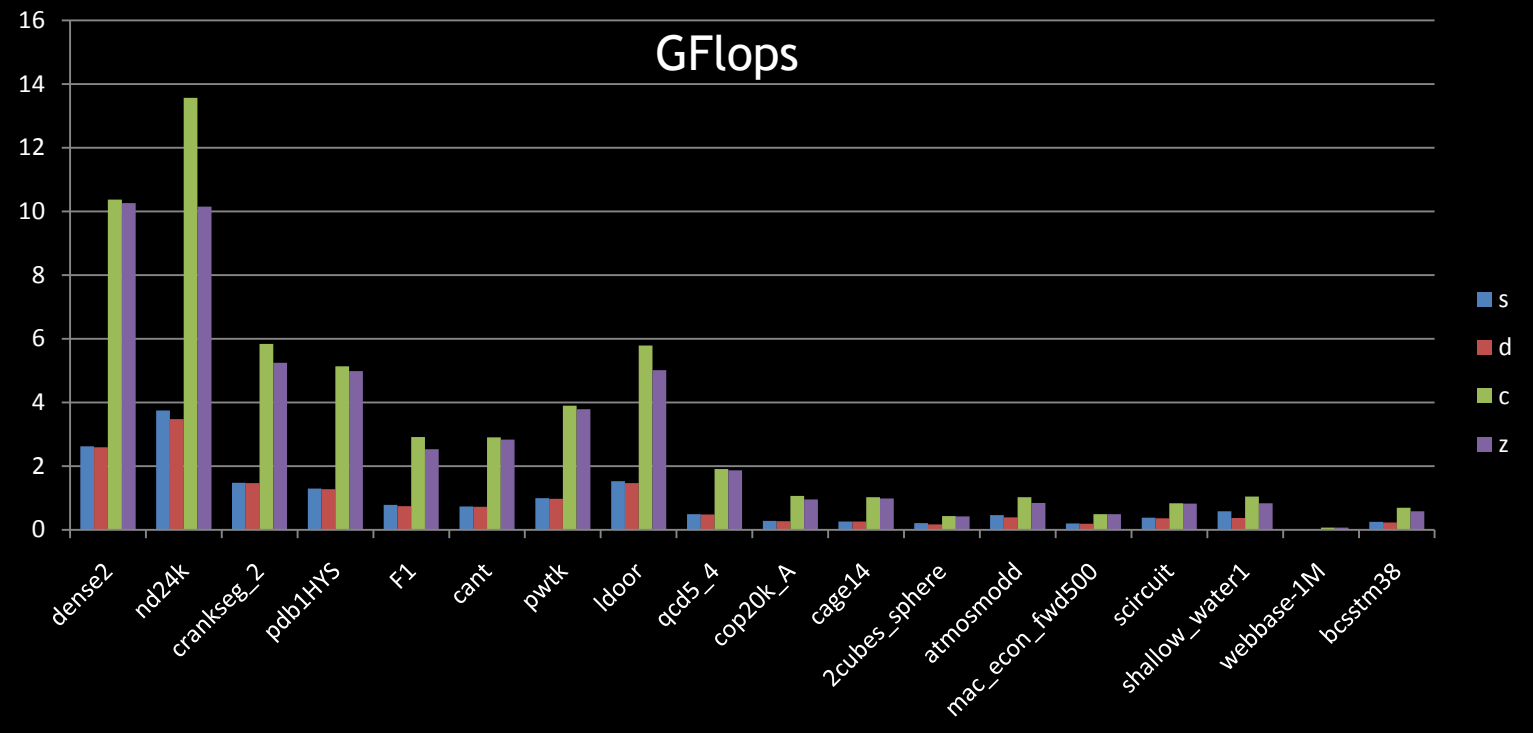
$$A = \begin{bmatrix} a_1^T \\ a_2^T \\ a_3^T \\ a_4^T \end{bmatrix} = \begin{bmatrix} 1.0 & & & \\ 2.0 & 3.0 & & \\ & & 4.0 & \\ 5.0 & & 6.0 & 7.0 \end{bmatrix}$$

$$y = \begin{bmatrix} a_1 & a_2 & a_3 & a_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

$$\begin{bmatrix} 1.0 & 2.0 & & 5.0 \\ & 3.0 & & \\ & & 4.0 & 6.0 \\ & & & 7.0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

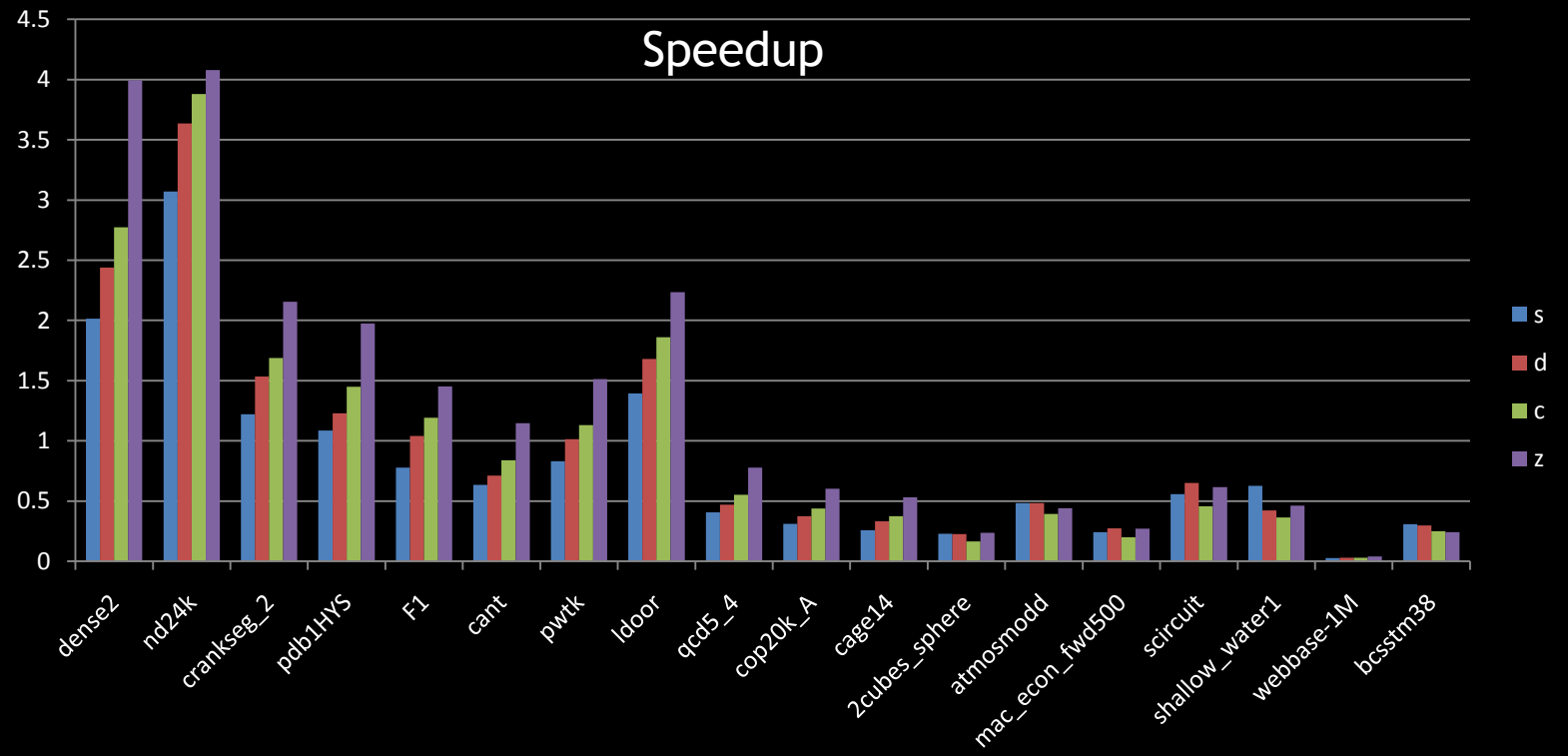
$$y=a_1*x_1+a_2*x_2+a_3*x_3+a_4*x_4$$

Performance of csrmv transpose



*NVIDIA C2050, ECC on

Performance of csrmv transpose



*NVIDIA C2050, ECC on
*MKL 10.2.3 , Core™ i7 @ 3.07GHz

Detailed study of csrmm transpose

- How about multiplication by multiple vectors (tall-matrices)
  - ✓ Again use similar approach for processing the sparse matrix

Detailed study of csrmm transpose

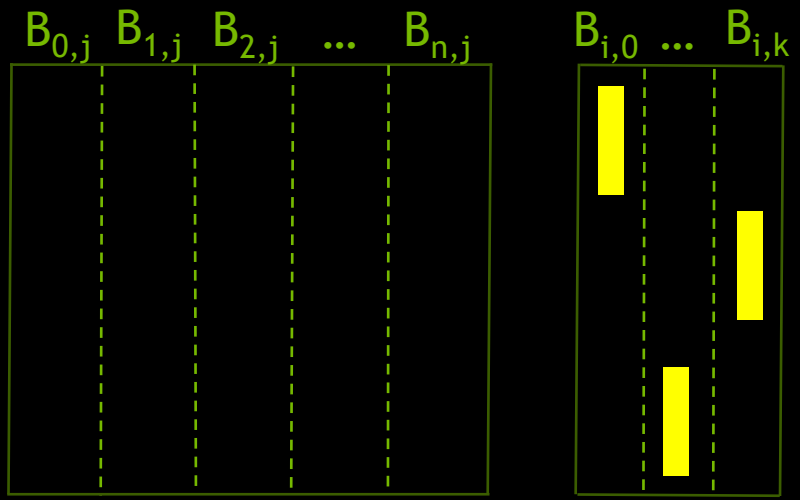
- How about multiplication by multiple vectors (tall-matrices)
  - ✓ Again use similar approach for processing the sparse matrix
  - ✓ Have a 2D grid of blocks (with second dimension processing columns of the tall-matrix)

Detailed study of csrmm transpose

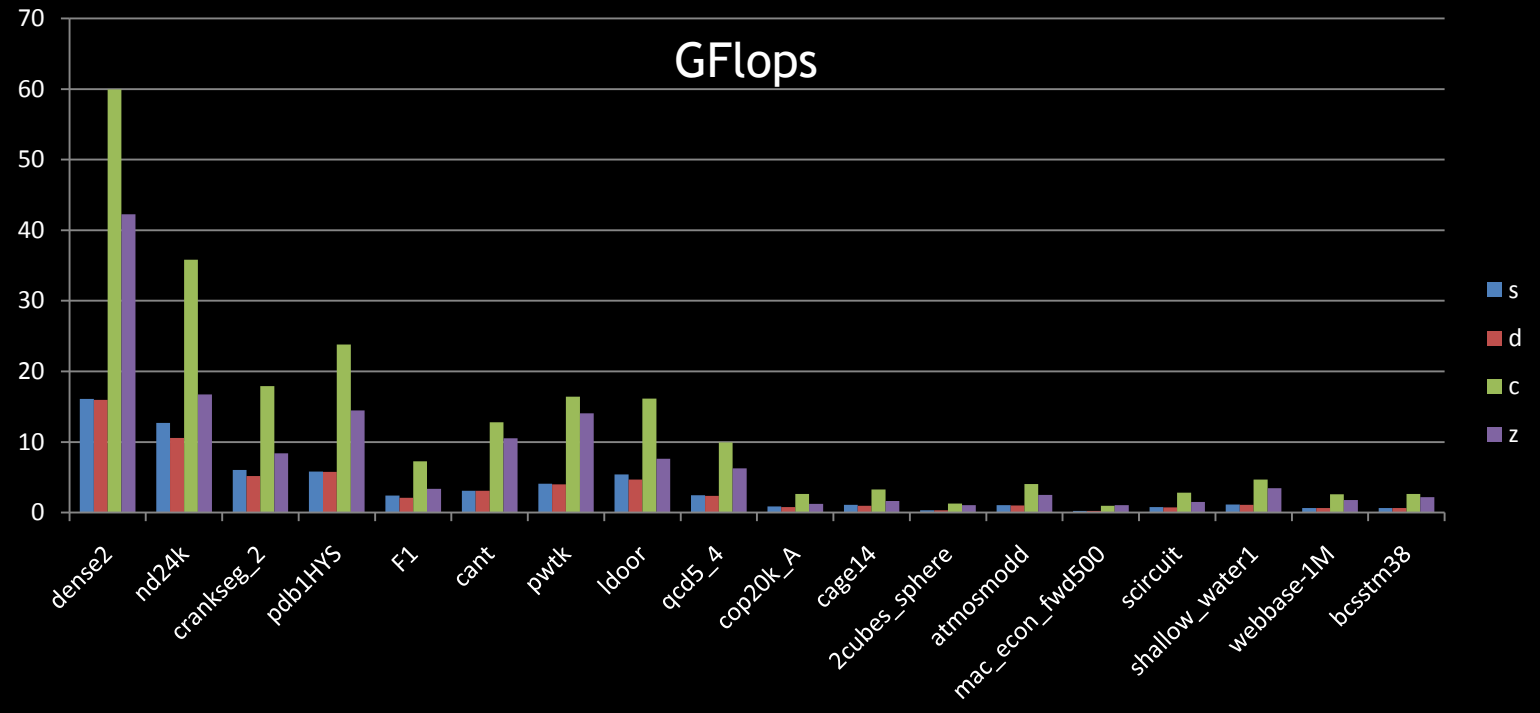
- How about multiplication by multiple vectors (tall-matrices)
  - ✓ Again use similar approach for processing the sparse matrix
  - ✓ Have a 2D grid of blocks (with second dimension processing columns of the tall-matrix)
  - ✓ Each block processes multiple columns (to minimize reading sparse matrix multiple times

Detailed study of csrmm transpose

- How about multiplication by multiple vectors (tall-matrices)
  - ✓ Again use similar approach for processing the sparse matrix
  - ✓ Have a 2D grid of blocks (with second dimension processing columns of the tall-matrix)
  - ✓ Each block processes multiple columns (to minimize reading sparse matrix multiple times)
  - ✓ Lock multiple elements at once, instead of per element atomics

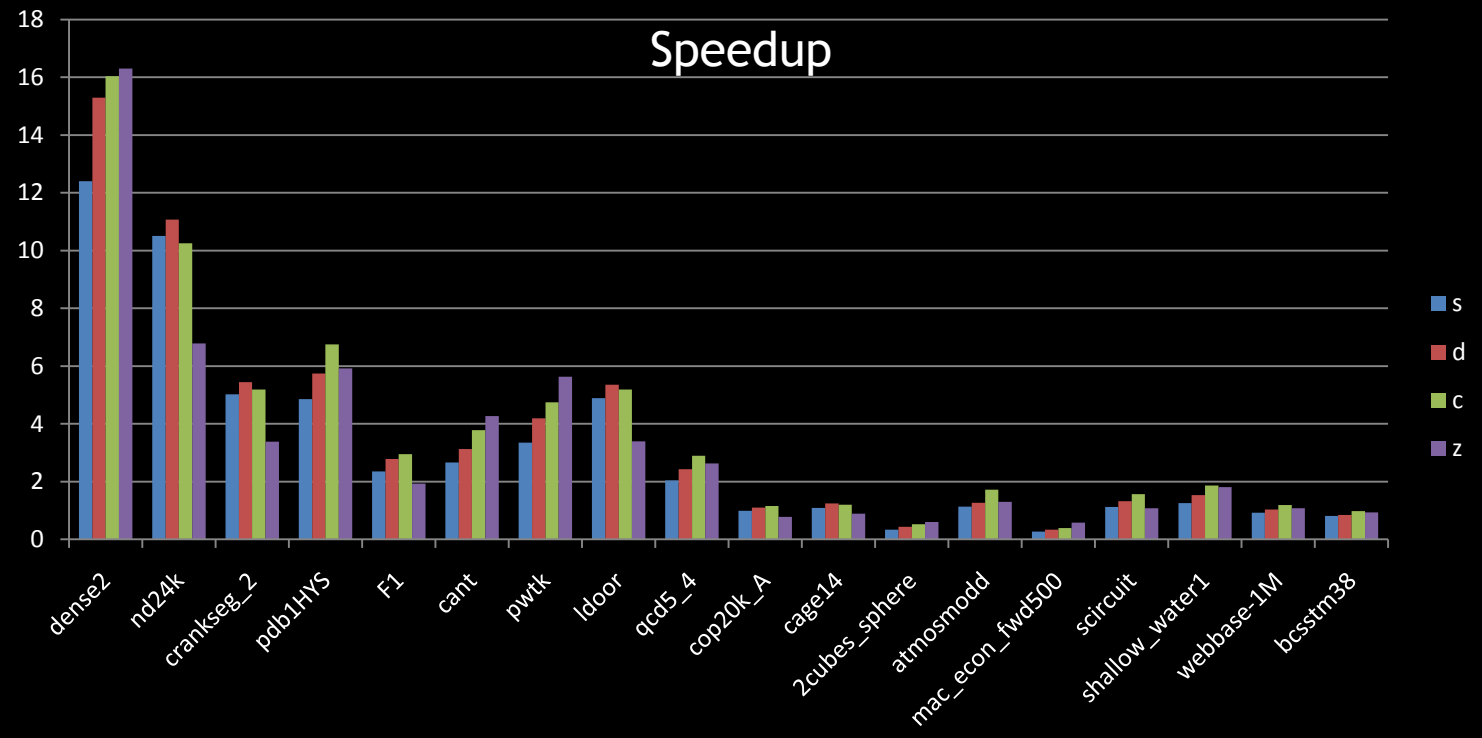


Performance of csrmm transpose



*NVIDIA C2050, ECC on

Performance of csrmm transpose



*NVIDIA C2050, ECC on
*MKL 10.2.3 , Core™ i7 @ 3.07GHz

CUSPARSE and other Libraries

- CUSPARSE
  - ✓ C interface, CUDA Toolkit
  - ✓ Sparse linear algebra (matrix multiplication, format conversions, triangular solve*) …
- CUBLAS
  - ✓ Dense linear algebra, CUDA Toolkit
- CUSP
  - ✓ C++ interface, Google Code
  - ✓ Iterative solvers, preconditioners, graph algorithms, …
- Thrust
  - ✓ Standard Template Library (STL) like interface, Google Code
  - ✓ Search, sort, reduce, …

*: future work

CUSPARSE and other Libraries

- CUSPARSE
  - ✓ C interface, CUDA Toolkit
  - ✓ Sparse linear algebra (matrix multiplication, format conversions, triangular solve*) …
- CUBLAS
  - ✓ Dense linear algebra, CUDA Toolkit
- CUSP
  - ✓ C++ interface, Google Code
  - ✓ Iterative solvers, preconditioners, graph algorithms, …
- Thrust
  - ✓ Standard Template Library (STL) like interface, Google Code
  - ✓ Search, sort, reduce, …

*: future work

CUSPARSE and other Libraries

- CUSPARSE
  - ✓ C interface, CUDA Toolkit
  - ✓ Sparse linear algebra (matrix multiplication, format conversions, triangular solve*) …
- CUBLAS
  - ✓ Dense linear algebra, CUDA Toolkit
- CUSP
  - ✓ C++ interface, Google Code
  - ✓ Iterative solvers, preconditioners, graph algorithms, …
- Thrust
  - ✓ Standard Template Library (STL) like interface, Google Code
  - ✓ Search, sort, reduce, …

*: future work

CUSPARSE and other Libraries

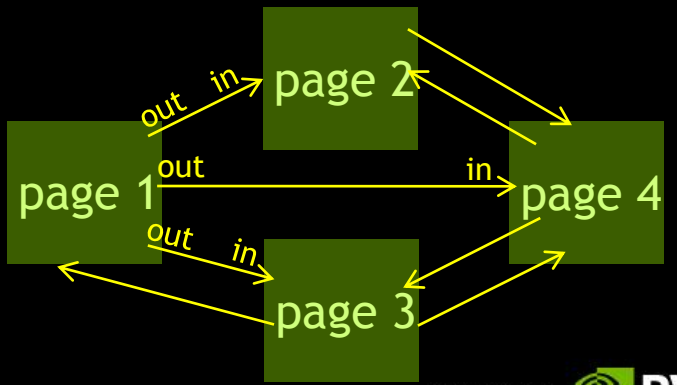
- CUSPARSE
  - ✓ C interface, CUDA Toolkit
  - ✓ Sparse linear algebra (matrix multiplication, format conversions, triangular solve*) …
- CUBLAS
  - ✓ Dense linear algebra, CUDA Toolkit
- CUSP
  - ✓ C++ interface, Google Code
  - ✓ Iterative solvers, preconditioners, graph algorithms, …
- Thrust
  - ✓ Standard Template Library (STL) like interface, Google Code
  - ✓ Search, sort, reduce, …

*: future work

Google PageRank

- Find the rank (importance) of a webpage
  - ✓ Your rank r_i is proportional to the # and rank of webpages with (in-) links to you
  - ✓ $r_i = \text{sum}(r_j / n_j)$, where n_j is the # of (out-) links of webpage j
- This problem can be formulated as an eigenvalue problem
  - ✓ with column stochastic matrix (nonnegative elements & sum of col. elements =1)
  - ✓ we need to obtain the largest eigenvalue (=1) and the corresponding eigenvector (which will give us the ranks r_i)

$$\begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \end{bmatrix} = \begin{bmatrix} & & 1/2 & \\ 1/3 & & & 1/2 \\ 1/3 & & & 1/2 \\ 1/3 & 1 & 1/2 & \end{bmatrix} \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \end{bmatrix}$$



Google PageRank

- Power Method
  - ✓ Finds the largest eigenvalue and the corresponding eigenvector of a matrix
  - ✓ Main ideas
    - Let the eigenvalue problem to be solved be

$$A x = \lambda x$$

    - Assume

$$\lambda_1 =< \ldots =< \lambda_{n-1} < \lambda_n$$

    - Then, notice (as $k \rightarrow \infty$)

$A^k x =$

Google PageRank

- Power Method
  - ✓ Finds the largest eigenvalue and the corresponding eigenvector of a matrix
  - ✓ Main ideas
    - Let the eigenvalue problem to be solved be

$$A x = \lambda x$$

    - Assume

$$\lambda_1 =< \ldots =< \lambda_{n-1} < \lambda_n$$

    - Then, notice (as $k \rightarrow \infty$)

$$A^k x = A^k (\alpha_n u_n + \alpha_{n-1} u_{n-1} + \ldots + \alpha_1 u_1)$$

Google PageRank

- Power Method
  - ✓ Finds the largest eigenvalue and the corresponding eigenvector of a matrix
  - ✓ Main ideas
    - Let the eigenvalue problem to be solved be

$$A x = \lambda x$$

    - Assume

$$\lambda_1 =< \ldots =< \lambda_{n-1} < \lambda_n$$

    - Then, notice (as $k \rightarrow \infty$)

$$A^k x = A^k (\alpha_n u_n + \alpha_{n-1} u_{n-1} + \ldots + \alpha_1 u_1)$$
$$= \lambda_n^k \alpha_n u_n + \lambda_{n-1}^k \alpha_{n-1} u_{n-1} + \ldots + \lambda_1^k \alpha_1 u_1$$

Google PageRank

- Power Method
  - ✓ Finds the largest eigenvalue and the corresponding eigenvector of a matrix
  - ✓ Main ideas
    - Let the eigenvalue problem to be solved be

$$A x = \lambda x$$

    - Assume

$$\lambda_1 =< \ldots =< \lambda_{n-1} < \lambda_n$$

    - Then, notice (as $k \rightarrow \infty$)

$$A^k x = A^k (\alpha_n u_n + \alpha_{n-1} u_{n-1} + \ldots + \alpha_1 u_1)$$
$$= \lambda_n^k \alpha_n u_n + \lambda_{n-1}^k \alpha_{n-1} u_{n-1} + \ldots + \lambda_1^k \alpha_1 u_1$$
$$= \lambda_n^k [\alpha_n u_n + (\lambda_{n-1}/\lambda_n)^k \alpha_{n-1} u_{n-1} + \ldots]$$

Google PageRank

- Power Method
  - ✓ Finds the largest eigenvalue and the corresponding eigenvector of a matrix
  - ✓ Main ideas
    - Let the eigenvalue problem to be solved be

$$A x = \lambda x$$

    - Assume

$$\lambda_1 =< \ldots =< \lambda_{n-1} < \lambda_n$$

    - Then, notice (as $k \rightarrow \infty$)

$$A^k x = A^k (\alpha_n u_n + \alpha_{n-1} u_{n-1} + \ldots + \alpha_1 u_1)$$
$$= \lambda_n^k \alpha_n u_n + \lambda_{n-1}^k \alpha_{n-1} u_{n-1} + \ldots + \lambda_1^k \alpha_1 u_1$$
$$= \lambda_n^k [\alpha_n u_n + (\lambda_{n-1}/\lambda_n)^k \alpha_{n-1} u_{n-1} + \ldots]$$

$(\lambda_{n-1}/\lambda_n)^k \rightarrow 0$

Google PageRank

- Power Method
  - ✓ Finds the largest eigenvalue and the corresponding eigenvector of a matrix
  - ✓ Main ideas
    - Let the eigenvalue problem to be solved be

$$A x = \lambda x$$

    - Assume

$$\lambda_1 =< \ldots =< \lambda_{n-1} < \lambda_n$$

    - Then, notice (as $k \rightarrow \infty$)

$$A^k x = A^k (\alpha_n u_n + \alpha_{n-1} u_{n-1} + \ldots + \alpha_1 u_1)$$
$$= \lambda_n^k \alpha_n u_n + \lambda_{n-1}^k \alpha_{n-1} u_{n-1} + \ldots + \lambda_1^k \alpha_1 u_1$$
$$= \lambda_n^k [\alpha_n u_n + (\lambda_{n-1}/\lambda_n)^k \alpha_{n-1} u_{n-1} + \ldots]$$

where $(\lambda_{n-1}/\lambda_n)^k \rightarrow 0$

    - Convergence is proportional to ratio $\lambda_{n-1} / \lambda_n$

Google PageRank

- Power Method
  - ✓ Finds the largest eigenvalue and the corresponding eigenvector of a matrix
  - ✓ Main ideas
    - Let the eigenvalue problem to be solved be

$$A x = \lambda x$$

    - Assume

$$\lambda_1 =< \ldots =< \lambda_{n-1} < \lambda_n$$

    - Then, notice (as $k \rightarrow \infty$)

$$A^k x = A^k (\alpha_n u_n + \alpha_{n-1} u_{n-1} + \ldots + \alpha_1 u_1)$$
$$= \lambda_n^k \alpha_n u_n + \lambda_{n-1}^k \alpha_{n-1} u_{n-1} + \ldots + \lambda_1^k \alpha_1 u_1$$
$$= \lambda_n^k [\alpha_n u_n + (\lambda_{n-1}/\lambda_n)^k \alpha_{n-1} u_{n-1} + \ldots]$$

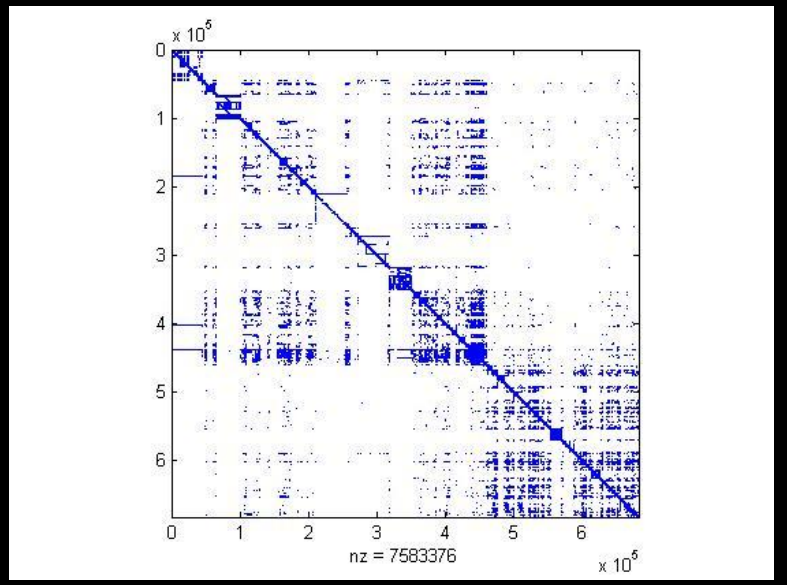
where $(\lambda_{n-1}/\lambda_n)^k \rightarrow 0$

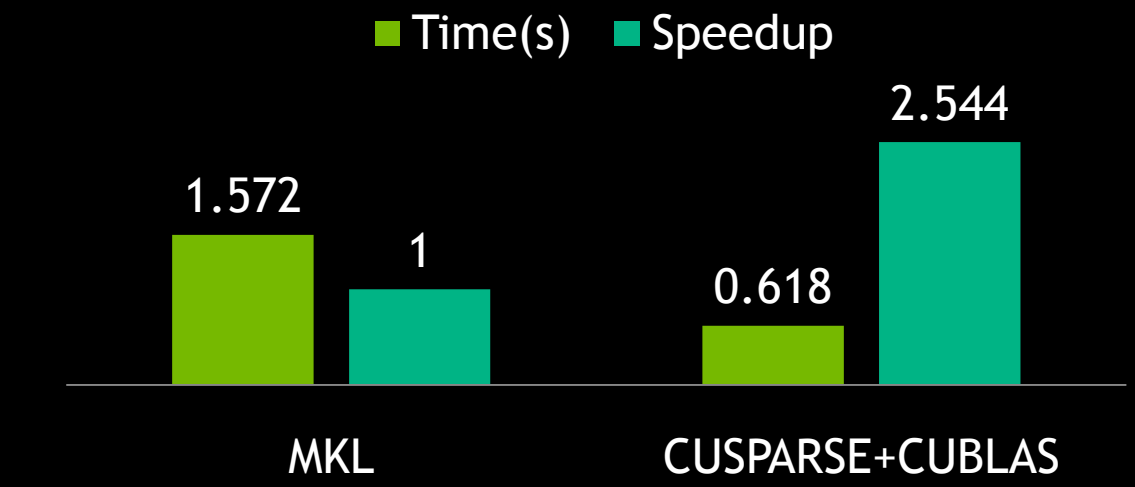
    - Convergence is proportional to ratio $\lambda_{n-1} / \lambda_n$

  - ✓ Key operation is sparse matrix-vector multiplication (csrmv)

Google PageRank

- Consider a realistic problem: Stanford-Berkeley (web connectivity matrix)





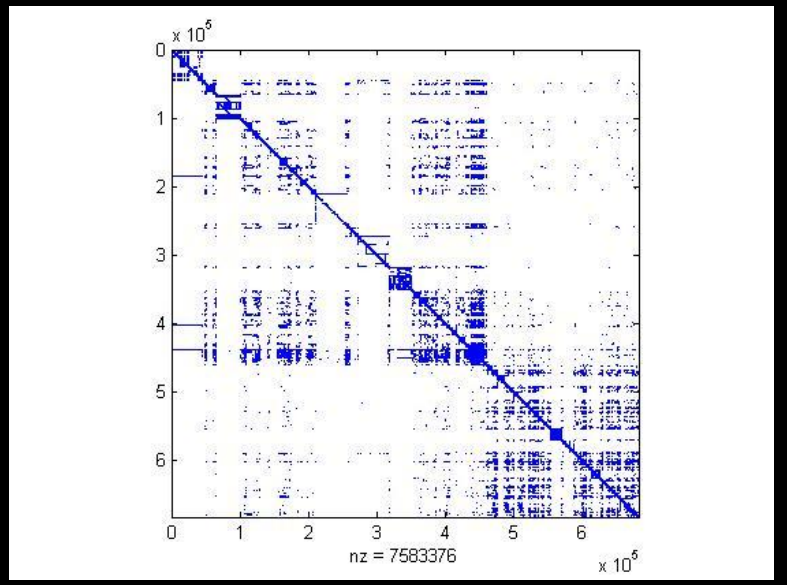
Double Precision, Stopping criteria: 54 iterations, 10^{-8}

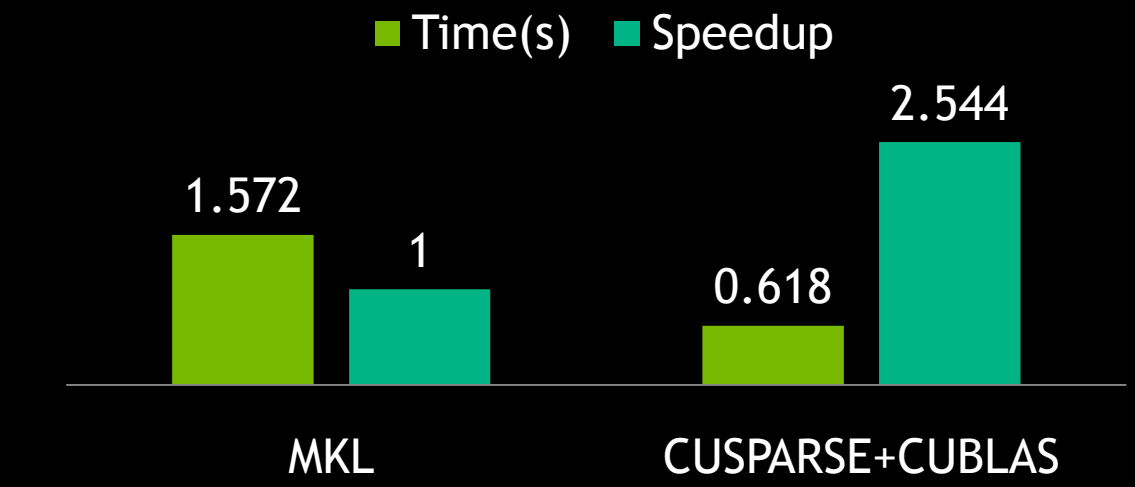
- Subspace Iteration
  - ✓ faster convergence
  - ✓ key operation is sparse matrix-tall-matrix multiplication (csrmm)

*NVIDIA C2050, ECC on
*MKL 10.2.3 , Core™ i7 @ 3.07GHz

Google PageRank

- Consider a realistic problem: Stanford-Berkeley (web connectivity matrix)





Double Precision, Stopping criteria: 54 iterations, 10^{-8}

- Subspace Iteration
  - ✓ faster convergence
  - ✓ key operation is sparse matrix-tall-matrix multiplication (csrmm)

2.5x speedup

*NVIDIA C2050, ECC on
*MKL 10.2.3 , Core™ i7 @ 3.07GHz

Atmospheric Modeling

- Solve 3D Navier-Stokes equations

$$\begin{cases} dV/dt = -RT \operatorname{grad} P + m\triangle V & \text{(momentum equation)} \\ dP/dt = -\ c_p/c_v \operatorname{div} V & \text{(continuity equation)} \\ dT/dt = -RT/c_v \operatorname{div} V & \text{(thermodynamic equation)} \end{cases}$$

for unknown velocity V, pressure logarithm P and temperature T.

- Numerical Solution
  - ✓ discretize using finite-difference or finite-element method
  - ✓ starting with initial conditions find unknowns at some given time
  - ✓ solving a large sparse linear system at every time step

Atmospheric Modeling

- Biconjugate Gradient Stabilized (BiCGStab) method
  - ✓ Krylov subspace iterative method

Atmospheric Modeling

- Biconjugate Gradient Stabilized (BiCGStab) method
  - ✓ Krylov subspace iterative method
  - ✓ Designed for nonsymmetric linear systems: Ax=f

Atmospheric Modeling

- Biconjugate Gradient Stabilized (BiCGStab) method
  - ✓ Krylov subspace iterative method
  - ✓ Designed for nonsymmetric linear systems: Ax=f

```
for(i=0; i<maxit; i++){
        \rhop=\rho
        \rho  = r^T r
        if (i > 0){
                \beta = (\rho / rhop) (\alpha / \omega)
                p       = r + \beta (p – \omega v)
        }
        v=A p
        \alpha = \rho / r^T v
        s = r – \alpha v
        t = A s
        \omega = t^T s / t^T t
        x = x + \alpha p + \omega s
        r = s - \omega t
}
```

Atmospheric Modeling

- Biconjugate Gradient Stabilized (BiCGStab) method
  - ✓ Krylov subspace iterative method
  - ✓ Designed for nonsymmetric linear systems: Ax=f

```
for(i=0; i<maxit; i++){
        \rhop=\rho
        \rho  = r^T r
        if (i > 0){
                \beta = (\rho / rhop) (\alpha / \omega)
                p     = r + \beta (p – \omega v)
        }
        v=A p
        \alpha = \rho / r^T v
        s = r – \alpha v
        t = A s
        \omega = t^T s / t^T t
        x = x + \alpha p + \omega s
        r = s - \omega t
}
```

dot product (CUBLAS)

Atmospheric Modeling

- Biconjugate Gradient Stabilized (BiCGStab) method
  - ✓ Krylov subspace iterative method
  - ✓ Designed for nonsymmetric linear systems: Ax=f

```
for(i=0; i<maxit; i++){
        \rhop=\rho
        \rho  = r^T r
        if (i > 0){
                \beta = (\rho / rhop) (\alpha / \omega)
                p     = r + \beta (p – \omega v)
        }
        v=A p
        \alpha = \rho / r^T v
        s = r – \alpha v
        t = A s
        \omega = t^T s / t^T t
        x = x + \alpha p + \omega s
        r = s - \omega t
}
```

dot product (CUBLAS)

scaled vector add (CUBLAS)

Atmospheric Modeling

- Biconjugate Gradient Stabilized (BiCGStab) method
  - ✓ Krylov subspace iterative method
  - ✓ Designed for nonsymmetric linear systems: Ax=f

```
for(i=0; i<maxit; i++){
        \rhop=\rho
        \rho  = r^T r
        if (i > 0){
                \beta = (\rho / rhop) (\alpha / \omega)
                p     = r + \beta (p – \omega v)
        }
        v= A p
        \alpha = \rho / r^T v
        s = r – \alpha v
        t = A s
        \omega = t^T s / t^T t
        x = x + \alpha p + \omega s
        r = s - \omega t
}
```

dot product (CUBLAS)

scaled vector add (CUBLAS)

key operation

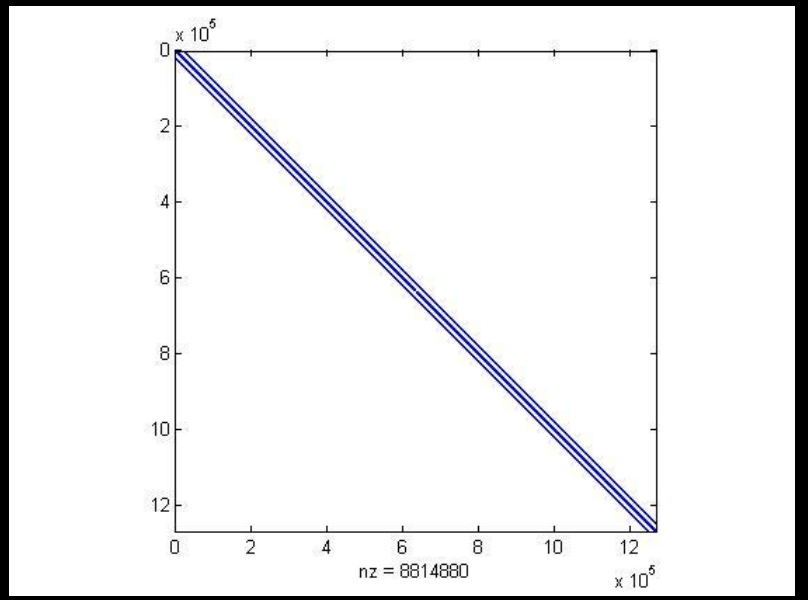
matrix-vector multiply (CUSPARSE)

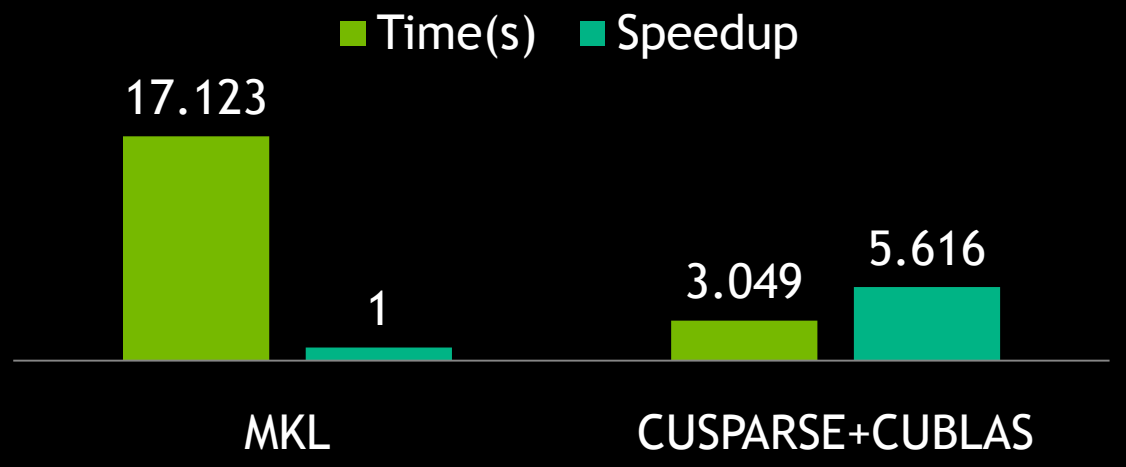
key operation

matrix-vector multiply (CUSPARSE)

Atmospheric Modeling

- Consider a realistic problem: AtmosModD (finite-difference discretization)





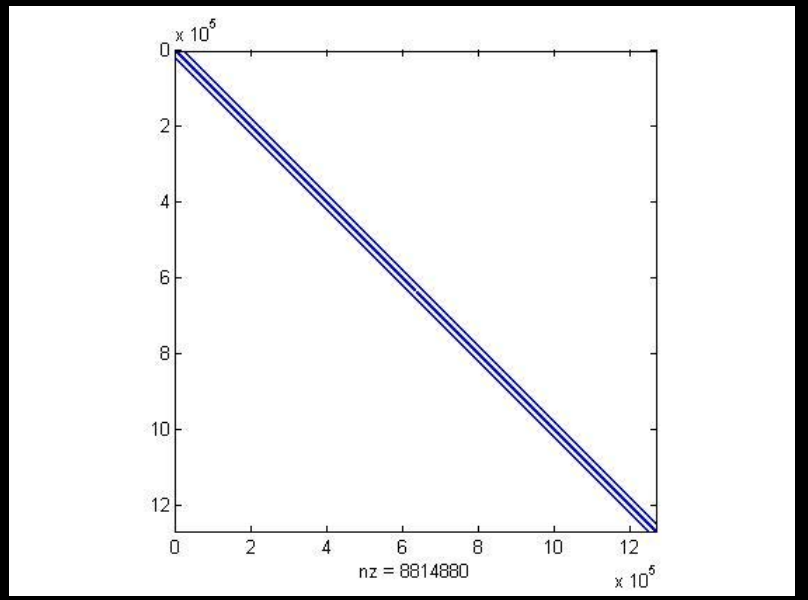
Double Precision, Stopping criteria: 256 iterations, 10^{-1}

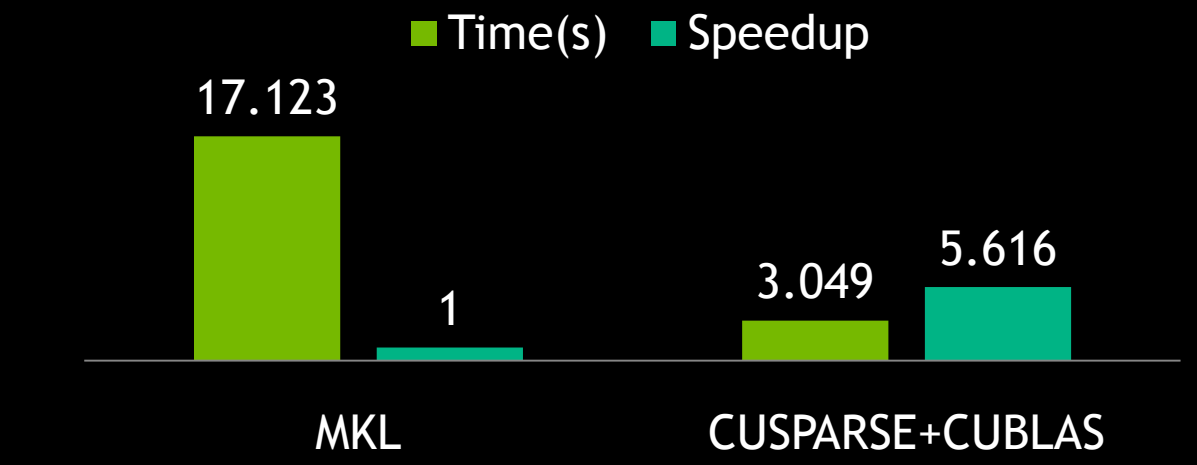
- Block Krylov subspace methods
  - ✓ faster convergence
  - ✓ key operation is sparse matrix-tall-matrix multiplication (csrmm)

*NVIDIA C2050, ECC on
*MKL 10.2.3 , Core™ i7 @ 3.07GHz

Atmospheric Modeling

- Consider a realistic problem: AtmosModD (finite-difference discretization)





Double Precision, Stopping criteria: 256 iterations, 10^{-1}

- Block Krylov subspace methods
  - ✓ faster convergence
  - ✓ key operation is sparse matrix-tall-matrix multiplication (csrmm)

5.6x speedup

*NVIDIA C2050, ECC on
*MKL 10.2.3 , Core™ i7 @ 3.07GHz

Conclusion

- CUSPARSE Library
  - ✓ Set of Basic Linear Algebra Subroutines for Sparse Matrices
  - ✓ Compressed Sparse Row format
    + conversion to and from other formats
- Applications
  - ✓ Iterative solution of linear systems and eigenvalue problems
  - ✓ Energy exploration, physical simulations and life sciences
- Future Work
  - ✓ New sparse storage formats
  - ✓ Solution of triangular linear systems
  - ✓ Preconditioning

References

[1] N. Bell and M. Garland, “Efficient Sparse Matrix-Vector Multiplication on CUDA” NVIDIA Technical Report NVR-2008-004, 2008.

[2] S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, J. Demmel, “Optimization of Sparse Matrix-Vector Multiplication on Emerging Multicore Platforms”, Supercomputing (SC), 2007.

[3] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine and H. Van der Vorst, “Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods”, SIAM, Philadelphia, PA, 1994.

[4] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe and H. van der Vorst, “Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide”, SIAM, Philadelphia, PA, 2000

[5] The University of Florida Sparse Matrix Collection, http://www.cise.ufl.edu/research/sparse/matrices/

Thank you