Extend Anasazi eigensolvers for billion-node graphs on an array of commodity SSDs

Da Zheng, Randal Burns, Joshua Vogelstein, Alexander Szalay
Johns Hopkins University

Overview

- FlashEigen extends the Anasazi eigensolvers to store sparse matrices and dense matrices on commodity SSDs.
- Our SSD eigensolver achieves the performance comparable to the in-memory implementation in a large parallel machine when computing a small number of eigenvalues.
- Our solution can compute eigenvalues of billion-node sparse graphs in a single machine.

Motivation

Sequential read: 540 MB/s
Sequential write: 480 MB/s

Motivation

Sequential read: 12 GB/s
Sequential write: 10 GB/s

One order of magnitude slower than RAM

Up to 24×

Motivation

Sequential read: 9 GB/s

Sequential write: 6.6 GB/s

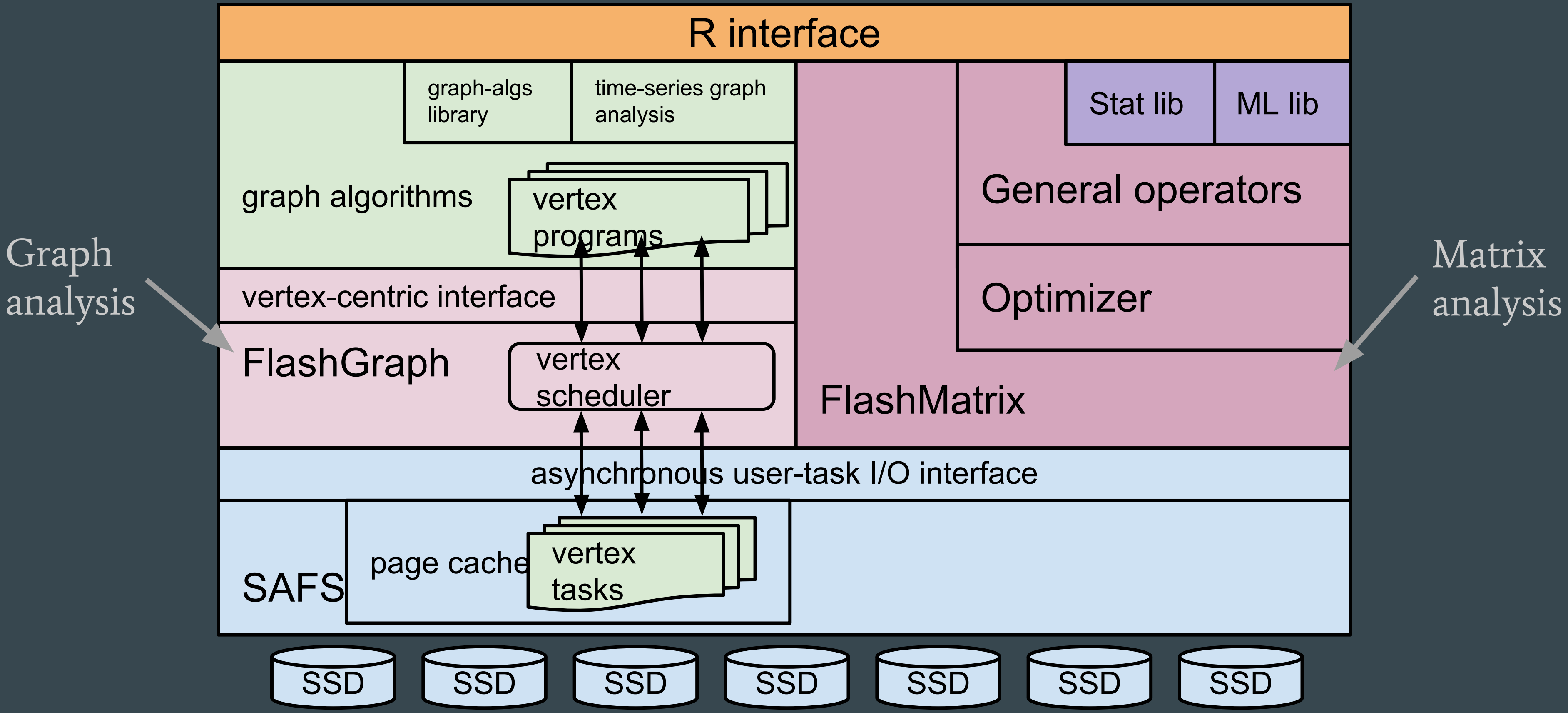
Can we replace RAM with SSDs?

- Target applications: large-scale data analysis.
- Speed vs. Scalability vs. Cost

Goals:

- Scalability >= 10
- Cost ≈ 10%
- Speed ≈ 50%

The full picture



We need an eigensolver

- Why to choose the Anasazi framework?
  - Extreme flexibility:
    - User-defined sparse matrix multiplication
    - User-defined dense matrices
  - Block extension.
  - Multiple state-of-art eigensolvers.

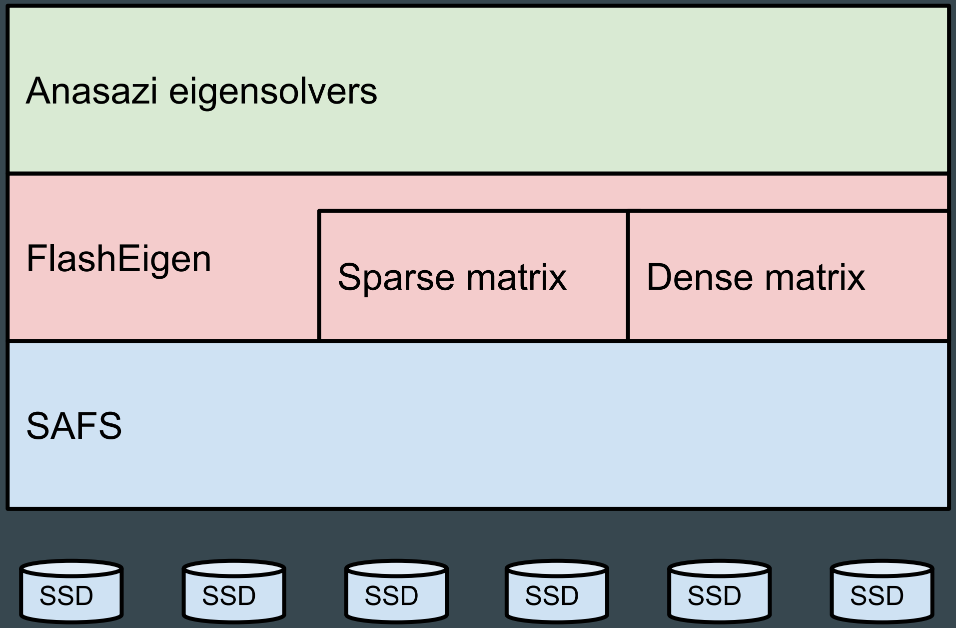
Target graphs

- Super sparse: |E| / |V| = 10~100
- Power-law distribution in vertex degree
- Nearly random vertex connection.
- Examples:
  - Social network graphs
  - Web graphs

=> The subspace requires roughly the same or larger storage size than the sparse matrix.

FlashEigen architecture

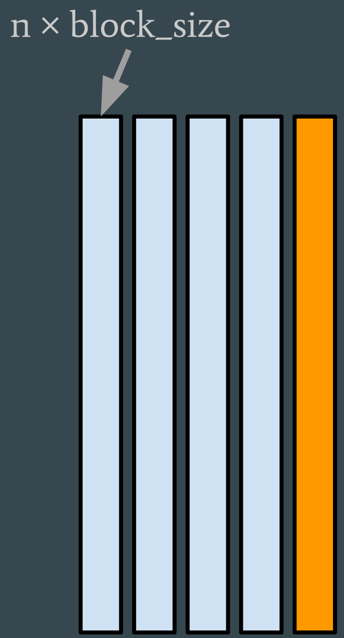
- Three layers:
  - SAFS
    - Deliver maximal I/O performance of SSDs
  - FlashEigen
    - A subset of FlashMatrix
      - Sparse matrix multiplication.
      - Dense matrix operations.
    - Implement Anasazi matrix operations
  - Anasazi
    - Unmodified code



Subspace

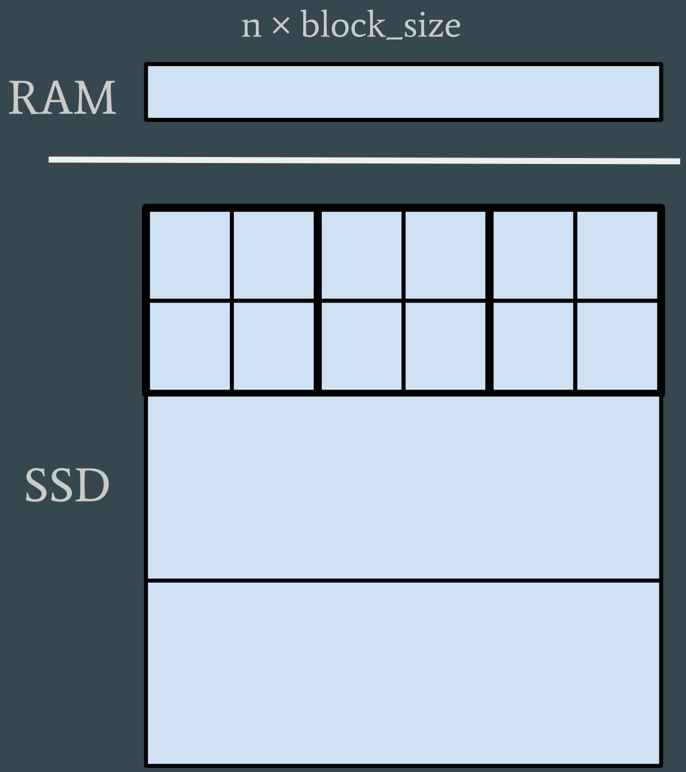
- The vector subspace storage >= the sparse matrix
  - Vectors are stored on SSDs.
  - Data is streamed to memory for computation => sequential I/O.
- Implement Anasazi::MultiVec
  - Vectors are groups into dense matrices (n × block_size).
  - Keep the most recent dense matrix in RAM to reduce I/O.
- The most I/O-intensive matrix operation:
  - Dense matrix multiplication for reorthogonalization.

MvTimesMatAddMv	MvAddMv	MvScale
MvTransMv	MvDot	MvNorm
SetBlock	MvRandom	MvInit



Sparse matrix

- Semi-external memory sparse matrix multiplication => sequential I/O
  - Sparse matrix (n × n) on SSDs.
  - Dense matrix (n × b) in RAM.
  - b has to be small.
- Implement Anasazi::OperatorTraits::apply().
- In-memory optimizations:
  - Cache blocking into small tiles to reduce CPU cache misses.
  - Group multiple tiles into super tiles based on the number of columns in dense matrices.



Supported eigensolvers in FlashEigen

- BlockKrylovSchur
- BlockDavidson
- LOBPCG
- We use BlockKrylovSchur for our eigenvalue problems:
  - The fastest in memory.
  - Generates the least I/O.
  - Use the least memory.

Graphs for performance evaluation

-

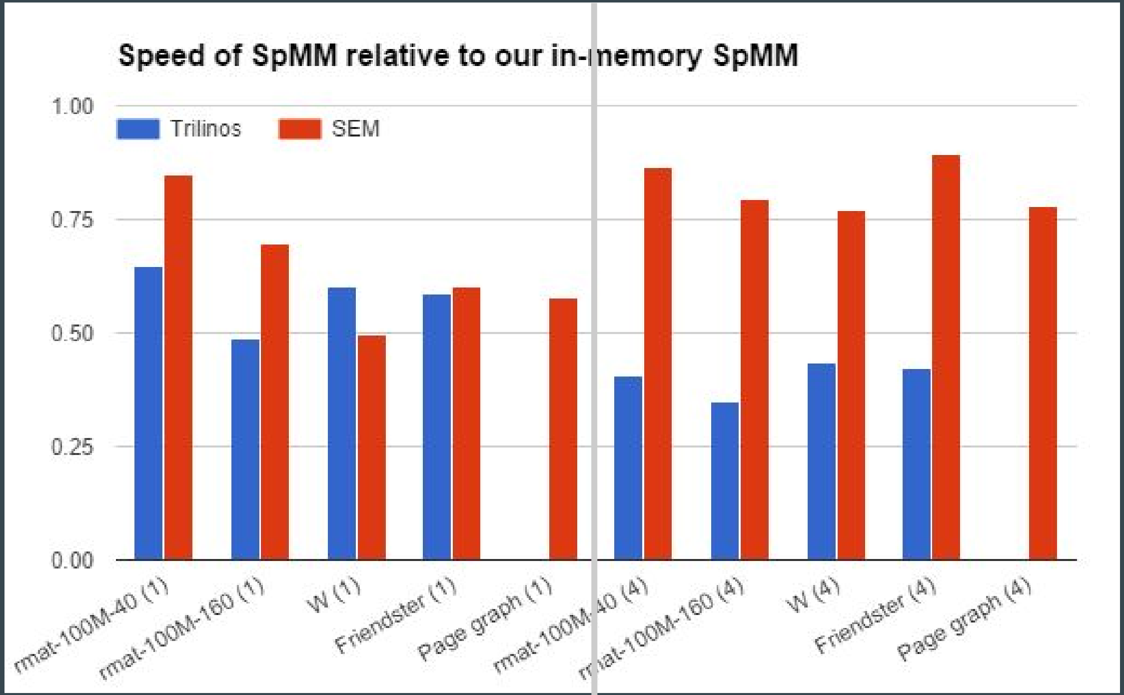
	# vertices	# edges
Friendster	65M	1.7B
KNN distance graph	62M	12B
RMat-100M-40	100M	3.7B
RMat-100M-160	100M	14B
Web page graph	3.4B	129B

Evaluation platform

- Dell PowerEdge R920
  - 4 Xeon CPU E7-4860 v2 @ 2.60GHz (48 cores)
  - 1TB DDR3-1600
- 24 OCZ Intrepid 3600 SATA SSD (10TB total)
- 3 LSI SAS 9300-8e host bus adapter
- The total cost: ~$50,000

Speed of sparse matrix multiplication (SpMM)

- Our semi-external memory (SEM) SpMM achieves at least 50% of our in-memory (IM) SpMM.
- Both our IM and SEM SpMM outperforms Trilinos, especially with 4 columns in the dense matrices.

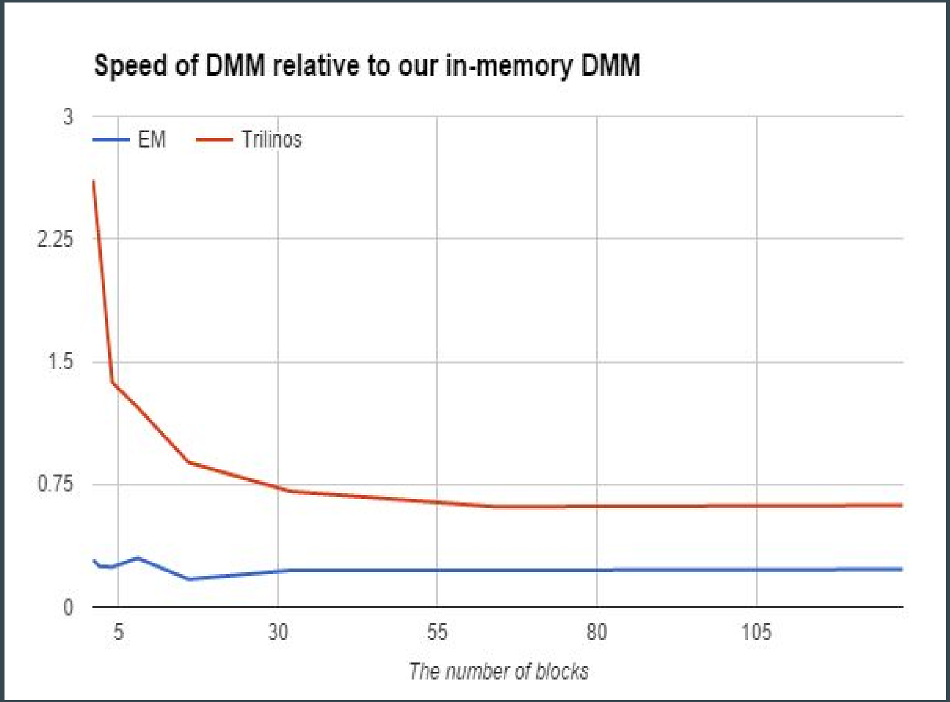


SpMV

Dense matrix has 4 columns

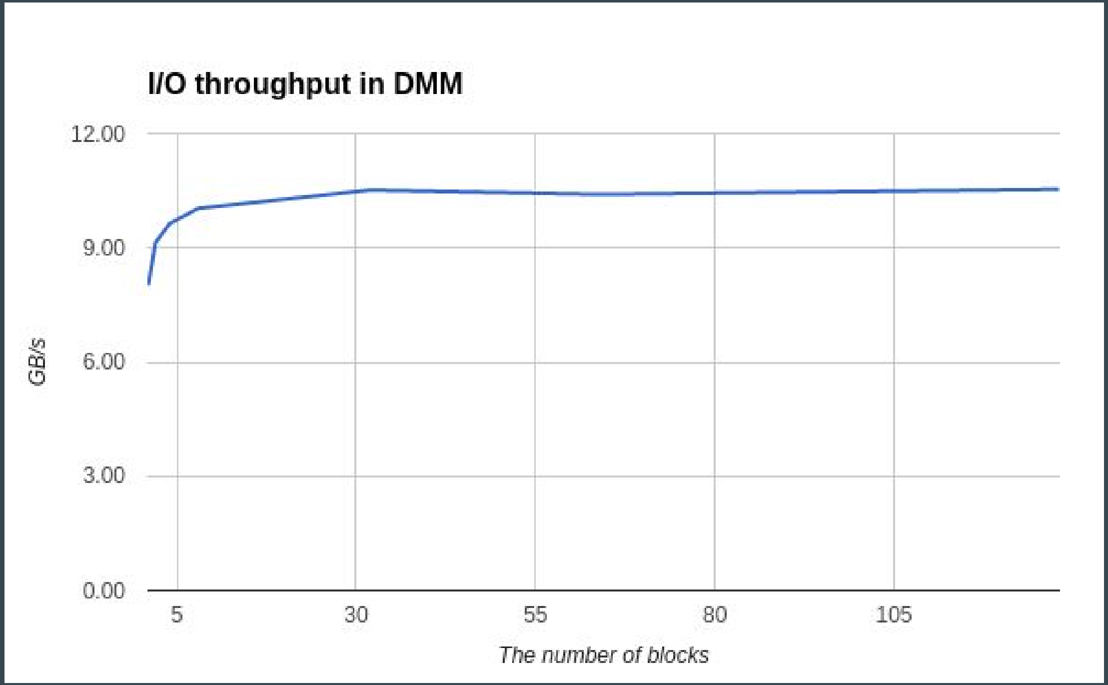
Speed of dense matrix multiplication (DMM)

- DMM for reorthogonalization
  - Block size = 4
  - Vary #blocks (1 - 128).
- Our EM DMM is only 25% of IM DMM.



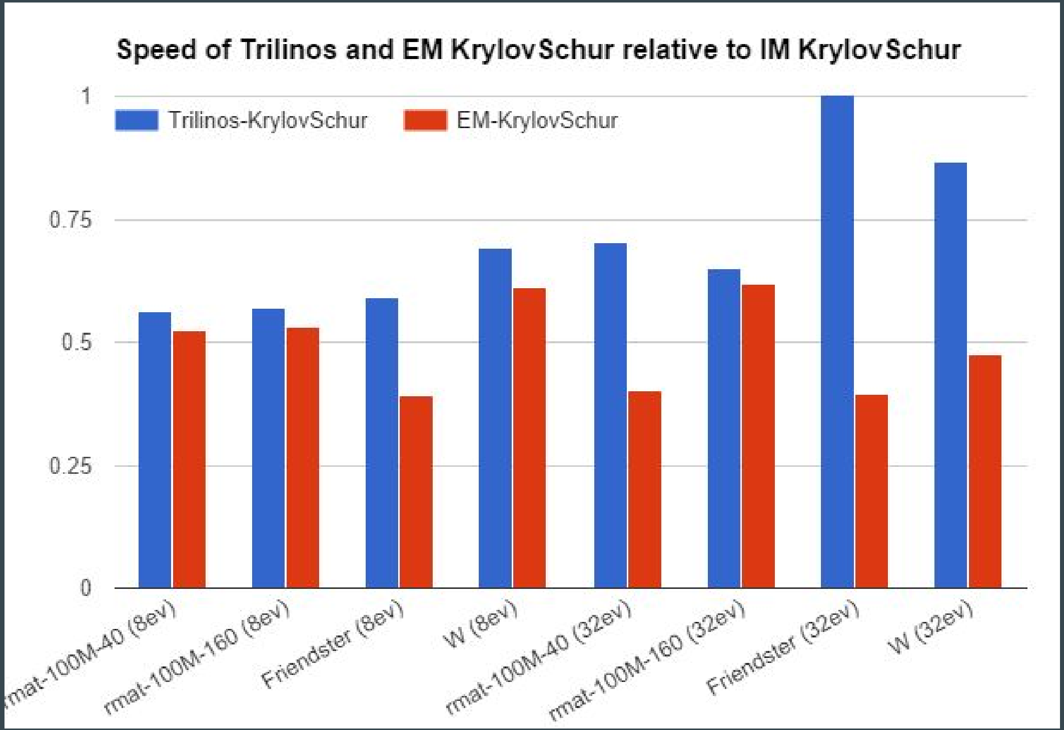
I/O throughput in EM DMM

- External-memory dense matrix multiplication is bottlenecked by SSDs.
    - Average I/O throughput is over 10GB/s.
    - The maximal I/O throughput of the hardware is 12GB/s.



Speed of eigensolvers

- EM KrylovSchur achieves 40%-60% speed of IM KrylovSchur.
- EM KrylovSchur has performance close to the Trilinos KrylovSchur.



Scalability of FlashEigen

- Page graph:

#eigenvalues	runtime	memory	read	write
8	4.2 hours	120GB	145TB	4TB
32	24 hours	120GB	922TB	11TB

- Average I/O throughput is 11GB/s.

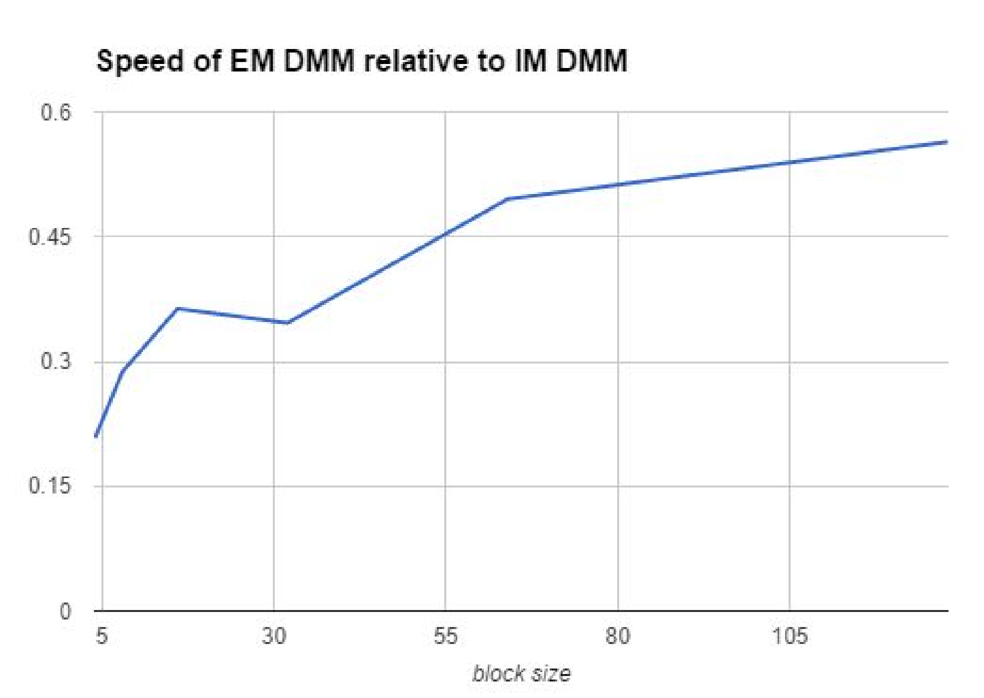
The story goes on (1)

- Good news:
  - Samsung enterprise SAS SSD (SM1635)
    - **Sequential read: 1400 MB/s**
    - Sequential write: 700 MB/s
    - Random read IOPS: 195K IOPS
    - Random write IOPS: 24K IOPS

=> 30GB/s with 24 SSDs?

The story goes on (2)

- DMM for reorthogonalization:
  - Subspace size: 128
  - The block size varies (4-128).
  - Computation increases by 32.
  - I/O increases by 2.
- Using a larger block size reduces the performance gap between IM and EM.
- Our solution works better for other eigensolvers such as BlockDavidson and LOBPCG.



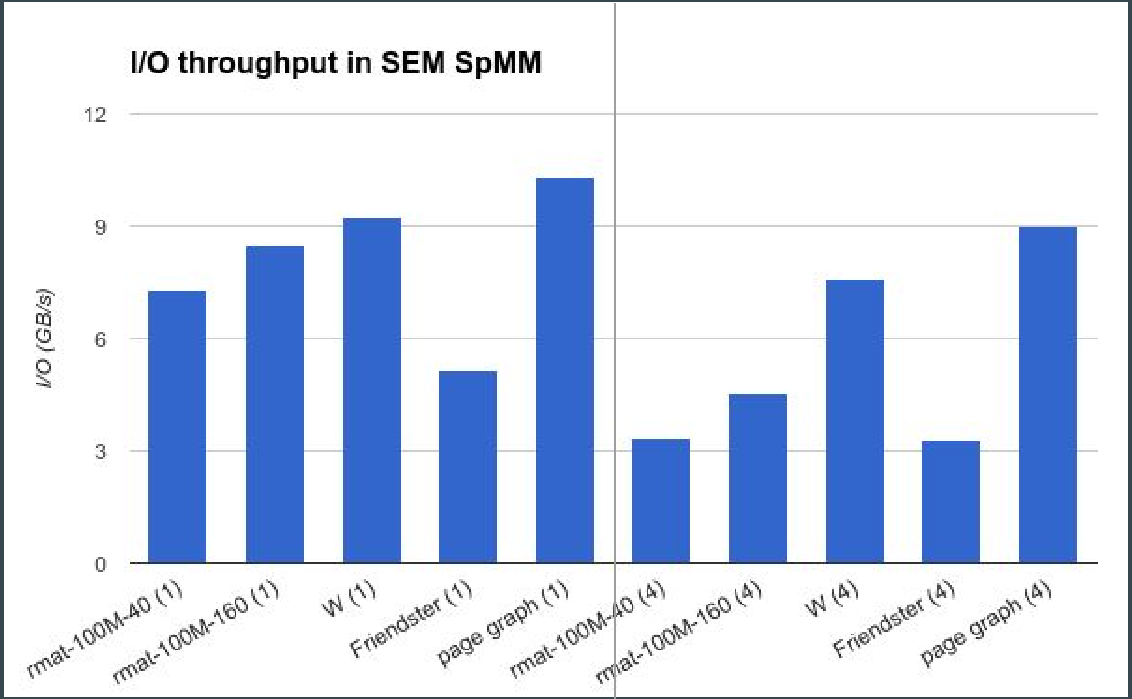
Conclusion

- The SSD-based eigensolver can have performance comparable to in-memory eigensolvers.
- For sparse graphs, SSDs are still the bottleneck, especially in dense matrix multiplication.

Thank you!
Da Zheng: dzheng5@jhu.edu
FlashEigen: https://github.com/icoming/FlashGraph

I/O throughput in SEM SpMM

- On some graphs, SpMV is bottlenecked by SSDs.



SpMV

SpMM with 4-col dense matrix