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 $\geq 10$
- Cost $\approx 10\%$
- Speed $\approx 50\%$
The full picture

SAFS
asynchronous user-task I/O interface
page cache
SSD
SSD
SSD
SSD
SSD
SSD
SSD
SSD

FlashGraph
graph algorithms
vertex-centric interface
vertex
vertex programs
vertex scheduler
FlashMatrix
General operators
Optimizer
Stat lib
ML lib
R interface
General operators
Optimizer
Stat lib
ML lib

Graph analysis
Matrix analysis

Graph analysis
Matrix analysis

FlashGraph
graph-algs library
time-series graph analysis

graph-algs library
time-series graph analysis

FlashMatrix
General operators
Optimizer
Stat lib
ML lib

R interface

SAFS
page cache
vertex
vertex tasks

SAFS
page cache
vertex
vertex tasks

SSD
SSD
SSD
SSD
SSD
SSD
SSD
SSD
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: $\frac{|E|}{|V|} = 10$~$100$
- 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 $\geq$ the sparse matrix
  - Vectors are stored on SSDs.
  - Data is streamed to memory for computation $\Rightarrow$ sequential I/O.
- Implement Anasazi::MultiVec
  - Vectors are grouped into dense matrices ($n \times \text{block}_\text{size}$).
  - Keep the most recent dense matrix in RAM to reduce I/O.
- The most I/O-intensive matrix operation:
  - Dense matrix multiplication for reorthogonalization.

<table>
<thead>
<tr>
<th>$\text{MvTimesMatAddMv}$</th>
<th>$\text{MvAddMv}$</th>
<th>$\text{MvScale}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{MvTransMv}$</td>
<td>$\text{MvDot}$</td>
<td>$\text{MvNorm}$</td>
</tr>
<tr>
<td>$\text{SetBlock}$</td>
<td>$\text{MvRandom}$</td>
<td>$\text{MvInit}$</td>
</tr>
</tbody>
</table>
Sparse matrix

- Semi-external memory sparse matrix multiplication => sequential I/O
  - Sparse matrix \((n \times n)\) on SSDs.
  - Dense matrix \((n \times b)\) 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

<table>
<thead>
<tr>
<th></th>
<th># vertices</th>
<th># edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Friendster</td>
<td>65M</td>
<td>1.7B</td>
</tr>
<tr>
<td>KNN distance graph</td>
<td>62M</td>
<td>12B</td>
</tr>
<tr>
<td>RMat-100M-40</td>
<td>100M</td>
<td>3.7B</td>
</tr>
<tr>
<td>RMat-100M-160</td>
<td>100M</td>
<td>14B</td>
</tr>
<tr>
<td>Web page graph</td>
<td>3.4B</td>
<td>129B</td>
</tr>
</tbody>
</table>
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
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.
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:

<table>
<thead>
<tr>
<th>#eigenvalues</th>
<th>runtime</th>
<th>memory</th>
<th>read</th>
<th>write</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>4.2 hours</td>
<td>120GB</td>
<td>145TB</td>
<td>4TB</td>
</tr>
<tr>
<td>32</td>
<td>24 hours</td>
<td>120GB</td>
<td>922TB</td>
<td>11TB</td>
</tr>
</tbody>
</table>

- 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.