Effective Parallel Computation of Eigenpairs to Detect Anomalies in Very Large Graphs

Michael M. Wolf, Benjamin A. Miller

SIAM PP14

18 February 2014
How do we address the data storage and compute challenges posed by the problem scales of interest to the DoD/IC community?
Big Data and HPC

Current approach: Map/Reduce

- Map \((<k_1, v_1> ) \rightarrow <k_2, v_2>\)
- Reduce \((k_2, \{<k_2, v_2>\}) \rightarrow v_3\)
- Each map-reduce step reads from and writes to disk

- Map/Reduce provides one way to deal with large problem sizes, but is too limited and too slow
  - Poorly suited for iterative sparse matrix and graph algorithms when fast runtime is essential
- Our approach uses High Performance Computing techniques to tackle big data
  - Leverage HPC sparse linear algebra packages (e.g., Trilinos)
Outline

- Big Data and High Performance Computing
- Anomaly Detection in Graphs
- Signal Processing for Graphs (SPG)
- Improving Sparse Matrix-Vector Multiplication (SpMV) Performance
- Improving Performance of Moving Average Filter
- Related Ongoing and Future Work
- Summary
Example Applications of Graph Analytics

**ISR**
- Graphs represent entities and relationships detected through multiple sources
- 1,000s – 1,000,000s tracks and locations
- GOAL: Identify anomalous patterns of life

**Social**
- Graphs represent relationships between individuals or documents
- 10,000s – 10,000,000s individual and interactions
- GOAL: Identify hidden social networks

**Cyber**
- Graphs represent communication patterns of computers on a network
- 1,000,000s – 1,000,000,000s network events
- GOAL: Detect cyber attacks or malicious software

Cross-Mission Challenge:
Detection of subtle patterns in massive multi-source noisy datasets
Example: Network Traffic Surrogate

Graph Statistics

- R-Mat
- Parameters derived from network traffic data
- 1024 vertices
Big Data Challenge: Activity Signatures

Graph Statistics

- R-Mat
- Parameters derived from network traffic data
- 1024 vertices
- Anomaly: 12 vertices
- Anomaly: 1% of graph (often smaller)

Challenge: Activity signature is typically a weak signal
Outline

• Big Data and High Performance Computing
• Anomaly Detection in Graphs
• Signal Processing for Graphs (SPG)
  • Improving Sparse Matrix-Vector Multiplication (SpMV) Performance
• Improving Performance of Moving Average Filter
• Related Ongoing and Future Work
• Summary
Statistical Detection Framework for Graphs

Graph Theory

Detection Theory

Develop fundamental graph signal processing concepts

Demonstrate in simulation

Apply to real data
Residuals Example: Anomalous Subgraph

- Residual graph represents the difference between the observed and expected.
- *Coordinated* vertices (subsets of vertices connected by edges with large edge weights) in residual graph will produce much stronger signal than uncoordinated vertices.

Detection framework is designed to detect coordinated deviations from the expected topology.
SPG Processing Chain

**Input**
- Graph
- No cue

**Output**
- Statistically anomalous subgraph(s)

**Steps:**
1. Temporal Integration
2. Graph Model Construction
3. Residual Decomposition
4. Component Selection
5. Anomaly Detection
6. Identification

**Dimensionality Reduction**
Anomaly Detection: Setup Phase

Detection Setup

1. Monte-Carlo simulations to generate density functions
2. ROC-curve generated from density function

H₀ – Null hypothesis, no signal
H₁ – Alternative hypothesis, signal
Anomaly Detection

**TEMPORAL INTEGRATION** → **GRAPH MODEL CONSTRUCTION** → **RESIDUAL DECOMPOSITION** → **COMPONENT SELECTION** → **ANOMALY DETECTION** → **IDENTIFICATION**

Test statistic calculated for observed graph:

\[
\chi^2_{max} = \max_{\theta} \chi^2 \left( \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}^T \right)
\]

**Threshold**

- **H₀** – Null hypothesis, no signal
- **H₁** – Alternative hypothesis, signal

Test statistic value significantly larger than test statistic value threshold corresponding to 1% false alarm rate
Detection Methods, Effectiveness, and Cost

Notional Comparison of Power and Effectiveness

- More powerful methods require more computation
- For detection of subtle anomalies, need to calculate 100s of eigenvectors fast

Eigenvectors $L_1$ Norms

$\chi^2$ in 2 Principal Components

Spectral Norm

$\sigma_1, \lambda_1$

Detection Power

Computation Cost

SPCA = Sparse Principal Component Analysis

$O((|E| r^2 + |V| r^3) h)^*$ to compute $r$ eigenvectors
Computational Focus: Dimensionality Reduction

- Dimensionality reduction dominates computation
- Eigen decomposition is key computational kernel
- Parallel implementation required for very large graph problems
  - Fit into memory
  - Minimize runtime

Need fast parallel eigensolvers
Eigenvalue Problems

\[ B = (A - E[A]) \]

Solve:
\[ Bx_i = \lambda_i x_i, i = 1, \ldots, m \]

<table>
<thead>
<tr>
<th>Modularity Matrix</th>
<th>Moving Average Filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ E[A_s] = \frac{k k^T}{2</td>
<td>e</td>
</tr>
</tbody>
</table>

- \(|e|\) – Number of edges in graph \(G(A)\)
- \(k\) – degree vector
- \(k_i = \text{degree}(v_i), v_i \in G(A)\)

- \(\bar{h} = \arg\min_h \left\| A_s(t) - \sum_{i=1}^{T} h_i A_s(t - i) \right\|_F\)
Modularity Matrix: Computation Breakdown

Matrix-vector multiplication is at the heart of eigensolver algorithms

Operator apply:

\[ Bx = A_s x - k \left( k^T x \right) / (2 |e|) \]

- Dense matrix-vector product: \( O(|V|^2) \)
- Sparse matrix-vector product: \( O(|e|) \)
- Dot product: \( O(|V|) \)
- Scalar-vector product: \( O(|V|) \)

\( Bx \) can be computed without storing \( B \) (modularity matrix)
Moving Average Filter: Computational Breakdown

Matrix-vector multiplication is at the heart of eigensolver algorithms

Operator:

\[ B(t) = A_s(t) - E[A_s(t)] \]

\[ E[A_s(t)] = \sum_{i=1}^{T} h_i A_s(t-i) \]

Since \( E[A(t)] \) is sparse, \( B(t) \) will be sparse

Key computational kernel is sparse matrix-dense vector multiplication
Parallel Implementation

• Using Anasazi (Trilinos) Eigensolver

• 64 bit global ordinals
  – Necessary for graphs with $2^{31}$ vertices or more

• User defined operators
  – Modularity matrix
  – Moving average filter
  – Apply defined efficiently for particular operator

• Block Krylov-Schur method
  – Symmetric
  – Eigenvalues with largest real component
  – Blocksize=1
Numerical Experiments

• Matrices
  – R-Mat (a=0.5, b=0.125, c=0.125, d=0.25)
    • Average nonzeros per row: 8
    • Number of rows: $2^{22}$ to $2^{32}$

• Two systems
  – LLGrid (MIT LL)
    • 274 compute nodes (8,768 cores)
    • Node: two 16-core AMD Opteron 6274 (2.2 GHz)
    • Network: 10 GB Ethernet
  – Hopper* (NERSC)
    • Cray XE6
    • 6,384 nodes (153,216 cores)
    • Node: two 12-core AMD 'MagnyCours' (2.1 GHz)
    • Network: 3D torus (Cray Gemini)

• Initially: 1D random row distribution (good load balance)

* This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.
Weak Scaling – Hopper*

Solved system for up to 4 billion vertex graph

**Runtime to Find 1st Eigenvector**

- **Hopper 1D**
- 4 billion vertices

**R-MAT, 2^{18} vertices/core**

Modularity Matrix

1D random partitioning

---

* This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.
Strong Scaling Results

Scalability limited and runtime increases for large numbers of cores.

* This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.
Finding Multiple Eigenvectors – LLGrid

Significant increase in runtime when finding additional eigenvectors.

Time to find 1, 2, 10, 100 eigenvalues/vectors

- 1 eigenvector
- 2 eigenvectors
- 10 eigenvectors
- 100 eigenvectors

R-MAT, $2^{23}$ vertices
Modularity Matrix

1D random partitioning

LLGrid system
Outline

• Big Data and High Performance Computing
• Anomaly Detection in Graphs
• Signal Processing for Graphs (SPG)
• Improving Sparse Matrix-Vector Multiplication (SpMV) Performance
• Improving Performance of Moving Average Filter
• Summary
Sparse Matrix-Vector Multiplication

- Sparse matrix-dense vector multiplication (SpMV) key computational kernel in eigensolver
- Performance of SpMV challenging for matrices resulting from power-law graphs
  - Load imbalance
  - Irregular communication
  - Little data locality
- Important to improve performance of SpMV
SpMV Strong Scaling -- LLGrid

R-Mat, 2^{23} vertices
Modularity Matrix

1D random partitioning

Scalability limited and runtime increases for large numbers of cores
Data Partitioning to Improve Parallel Sparse Matrix-Dense Vector Multiplication

\[
\begin{bmatrix}
  y_1 \\
  y_2 \\
  y_3 \\
  y_4 \\
  y_5 \\
  y_6 \\
  y_7 \\
  y_8
\end{bmatrix}
= 
\begin{bmatrix}
  1 & 6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
  5 & 1 & 9 & 0 & 5 & 0 & 0 & 0 & 0 \\
  0 & 8 & 1 & 7 & 0 & 0 & 0 & 0 & 0 \\
  0 & 0 & 2 & 1 & 0 & 0 & 0 & 7 & 0 \\
  0 & 0 & 0 & 0 & 1 & 8 & 0 & 0 & 0 \\
  4 & 0 & 0 & 0 & 3 & 1 & 3 & 0 & 0 \\
  0 & 0 & 0 & 6 & 0 & 9 & 1 & 4 & 0 \\
  0 & 0 & 0 & 0 & 0 & 0 & 2 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
  1 \\
  2 \\
  4 \\
  3 \\
  1 \\
  4 \\
  2 \\
  1
\end{bmatrix}
\]

\[y = Ax\]

- Partition matrix nonzeros
- Partition vectors
Communication Pattern: 1D Block Partitioning

2D Finite Difference Matrix (9 point)

Number of Rows: $2^{23}$
Nonzeros/Row: 9

$\text{NNZ/process}$
- min: $1.17E+06$
- max: $1.18E+06$
- avg: $1.18E+06$
- max/avg: 1.00

$\# \text{Messages (Phase 1)}$
- total: 126
- max: 2

$\text{Volume (Phase 1)}$
- total: $2.58E+05$
- max: $4.10E+03$

$\text{Nice properties:}$
- Great load balance
- Small number of messages
- Low communication volume
Communication Pattern: 1D Block Partitioning

R-Mat (0.5, 0.125, 0.125, 0.25)

Number of Rows: $2^{23}$
Nonzeros/Row: 8

**NNZ/process**
- min: $1.88E+05$
- max: $4.00E+06$
- avg: $1.06E+06$
- max/avg: 3.78

**# Messages (Phase 1)**
- total: 4032
- max: 63

**Volume (Phase 1)**
- total: $4.02E+07$
- max: $1.48E+06$

**Challenges:**
- Poor load balance
- All-to-all communication

P=64

Source process

Destination process
Communication Pattern: 1D Random Partitioning

R-Mat (0.5, 0.125, 0.125, 0.25)

Number of Rows: $2^{23}$
Nonzeros/Row: 8

**NNZ/process**
- min: $1.05E+06$
- max: $1.07E+06$
- avg: $1.06E+06$
- max/avg: 1.01

**# Messages (Phase 1)**
- total: 4032
- max: 63

**Volume (Phase 1)**
- total: $5.48E+07$
- max: $8.62E+05$

**Nice properties:**
Great load balance

**Challenges:**
All-to-all communication

P=64
2D Partitioning

<table>
<thead>
<tr>
<th>Block/Cartesian</th>
<th>Mondriaan (Vastenhouw, Bisseling)</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Block/Cartesian Diagram]</td>
<td>![Mondriaan Diagram]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fine-grain (Catalyurek, Aykanat)</th>
<th>Nested-dissection (Boman, Wolf)*</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Fine-grain Diagram]</td>
<td>![Nested-dissection Diagram]</td>
</tr>
</tbody>
</table>

- More flexibility: no particular part for entire row or column
- More general sets of nonzeros assigned parts

Bounding Number of Messages with 2D Partitioning

• Use flexibility of 2D partitioning to bound number of messages
  – Distribute nonzeros in permuted 2D Cartesian block manner

• 2D Random (Cartesian) – (Hendrickson, et al., Bisseling, Yoo)
  – Block Cartesian with rows/columns randomly distributed
  – Cyclic striping to minimize number of messages

• 2D Cartesian (Hyper)graph
  – Replace random partitioning with hyper(graph) partitioning to minimize communication volume
Communication Pattern: 2D Random Partitioning Cartesian Blocks (2DR)

R-Mat (0.5, 0.125, 0.125, 0.25)

Number of Rows: $2^{23}$
Nonzeros/Row: 8

NNZ/process
- min: $1.04E+06$
- max: $1.05E+06$
- avg: $1.05E+06$
- max/avg: 1.01

# Messages (Phase 1)
- total: 448
- max: 7

Volume (Phase 1)
- total: $2.57E+07$
- max: $4.03E+05$

Nice properties:
- No all-to-all communication
- Total volume lower than 1DR

1DR = 1D Random
Communication Pattern: 2D Random Partitioning Cartesian Blocks (2DR)

R-Mat (0.5, 0.125, 0.125, 0.25)

Number of Rows: $2^{23}$
Nonzeros/Row: 8

NNZ/process
min: 1.04E+06
max: 1.05E+06
avg: 1.05E+06
max/avg: 1.01

# Messages (Phase 2)
total: 448
max: 7

Volume (Phase 2)
total: 2.57E+07
max: 4.03E+05

Nice properties:
No all-to-all communication
Total volume lower than 1DR

1DR = 1D Random
Communication Pattern: 2D Cartesian Hypergraph Partitioning

R-Mat (0.5, 0.125, 0.125, 0.25)

Number of Rows: $2^{23}$
Nonzeros/Row: 8

**NNZ/process**
- min: 5.88E+05
- max: 1.29E+06
- avg: 1.05E+06
- max/avg: 1.23

**# Messages (Phase 1)**
- total: 448
- max: 7

**Volume (Phase 1)**
- total: 2.33E+07
- max: 4.52E+05

**Nice properties:**
- No all-to-all communication
- Total volume lower than 2DR

**Challenges:**
- Imbalance worse than 2DR

2DR = 2D Random Cartesian
Communication Pattern: 2D Cartesian Hypergraph Partitioning

R-Mat (0.5, 0.125, 0.125, 0.25)

Number of Rows: $2^{23}$
Nonzeros/Row: 8

**NNZ/process**
- min: $5.88E+05$
- max: $1.29E+06$
- avg: $1.05E+06$
- max/avg: 1.23

**# Messages (Phase 2)**
- total: 448
- max: 7

**Volume (Phase 2)**
- total: $2.54E+07$
- max: $4.80E+05$

**Nice properties:**
- No all-to-all communication
- Total volume lower than 2DR

**Challenges:**
- Imbalance worse than 2DR

2DR = 2D Random Cartesian
Improved Results: SpMV – LLGrid

Time needed to compute 10 SpMV operations

R-Mat, $2^{23}$ vertices/rows

Simple 2D method shows improved scalability
Improved Results – LLGrid

Runtime to Find 1st Eigenvector

R-Mat, $2^{23}$ vertices/rows
Modularity Matrix

Simple 2D method shows improved scalability
Improved Results – NERSC Hopper*

Runtime to Find 1st Eigenvector

2D methods show improved scalability

* This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.
Challenge with Hypergraph/Graph Partitioning

- High partitioning cost of graph/hypergraph methods must be amortized by computing many SpMV operations
- Detection** requires at most 1000s of SpMV operations
- Expensive partitions need to be effective for multiple graphs

**L1 norm method: computing 100 eigenvectors

Time to Partition and Compute SpMV operations

NERSC Hopper*

~40,000 SpMVs

R-Mat, 2^{23} vertices
1024 cores

* This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.
• Key question: How long will a partition be effective?
• Initial experiment
  – Evolving R-Mat matrices: fixed number of rows, R-Mat parameters \((a,b,c,d)\)
  – Start with a given number of nonzeros \((|e_0|)\)
  – Iteratively add nonzeros until new number of nonzeros is reached \((|e_n|)\)
Results: Partitioning for Dynamic Graphs

- \( |e_0| = 0.5 |e_n| \)
- 2D hypergraph surprisingly effective as edges are added to graph

* This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.
Results: Partitioning for Dynamic Graphs

- $|e_0| = 0.3 |e_n|$
- 2D hypergraph surprisingly effective as edges are added to graph

* This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.
Outline

• Big Data and High Performance Computing
• Anomaly Detection in Graphs
• Signal Processing for Graphs (SPG)
• Improving Sparse Matrix-Vector Multiplication (SpMV) Performance
• Improving Performance of Moving Average Filter
• Summary
Moving Average Filter

\[
E[A_s(t)] = \sum_{i=1}^{T} h_i A_s(t - i)
\]

\[
B = (A - E[A])
\]

- Option 1: explicitly form expected graph model matrix each time step
  - Pro: Less computation (when nonzeros collide) than \( T \) SpMV ops
  - Pro: Less communication than \( T \) SpMV ops
  - Con: Very expensive (have to add and subtract matrices to form)

- Option 2: don’t explicitly form graph model matrix
  - Pro: Avoid expensive matrix formation
  - Con: Requires \( T \) SpMV ops (more communication, possibly more computation)

- Idea to improve option 2: fuse multiple SpMV operations
  - Perform communication once
Fusing SpMV operations can effectively reduce runtime.

R-MAT, $2^{20}$ vertices
Moving average filter ($T=10$)

\[ E[A_s(t)] = \sum_{i=1}^{T} h_i A_s(t - i) \]
Outline

• Big Data and High Performance Computing
• Anomaly Detection in Graphs
• Signal Processing for Graphs (SPG)
• Improving Sparse Matrix-Vector Multiplication (SpMV) Performance
• Improving Performance of Moving Average Filter
• Summary
Summary

•Outlined HPC approach to processing big data
  – Signal processing for graphs
  – Statistical framework for anomaly detection in graphs
•Key component is eigensolver for dimensionality reduction
•Solving eigensystems resulting from power law graphs challenging
  – Load imbalance
  – Poor data locality
•SpMV key computational kernel
  – 1D data partitioning limits performance due to all-to-all communication
  – 2D data partitioning can be used to improve scalability
•Dynamic graphs pose new computational challenges
  – New computational kernels may be necessary (e.g., fused sparse matrix-dense vector operations)
Acknowledgements

- Nicholas Arcolano (MITLL)
- Michelle Beard (MITLL)
- Nadya Bliss (ASU)
- Jeremy Kepner (MITLL)
- Dan Kimball (MITLL)
- Lisie Michel (MITLL)
- Sanjeev Mohindra (MITLL)
- Eddie Rutledge (MITLL)
- Scott Sawyer (MITLL)
- Matt Schmidt (MITLL)