Scaling Parallel Graph Analysis & Machine Learning using Sparse Matrix Operations

Aydın Buluç
Computational Research Division, LBNL
EECS Department, UC Berkeley

NERSC Data Seminar
May 25, 2018
Large Graphs in Scientific Discoveries

Matching in bipartite graphs: Permuting to heavy diagonal or block triangular form

Graph partitioning: Dynamic load balancing in parallel simulations

Problem size: as big as the sparse linear system to be solved or the simulation to be performed

Picture (left) credit: Sanders and Schulz
Large Graphs in Scientific Discoveries

Whole genome assembly

A Read Layout
R1: GACCTACA
R2: ACCTACAA
R3: CCTACAAG
R4: CTACAAGT
A: TACAAGTT
B: ACAAGTAA
C: CAAGTTAG
X: TACAAGTC
Y: ACAAGTCC
Z: CAAGTCCG

B Overlap Graph

C de Bruijn Graph
Vertices: k-mers

Vertices: reads

26 billion (8B of which are non-erroneous) unique k-mers (vertices) in the hexaploid wheat genome W7984 for k=51

Schatz et al. (2010) Perspective: Assembly of Large Genomes w/2nd-Gen Seq. Genome Res. (figure reference)
Sparse Matrices

“I observed that most of the coefficients in our matrices were zero; i.e., the nonzeros were ‘sparse’ in the matrix, and that typically the triangular matrices associated with the forward and back solution provided by Gaussian elimination would remain sparse if pivot elements were chosen with care”

- Harry Markowitz, describing the 1950s work on portfolio theory that won the 1990 Nobel Prize for Economics
Graphs in the language of matrices

- Sparse array representation => space efficient
- Sparse matrix-matrix multiplication => work efficient
- Three possible levels of parallelism: searches, vertices, edges
- Highly-parallel implementation for Betweenness Centrality*

*: A measure of influence in graphs, based on shortest paths
Graph coarsening via sparse matrix-matrix products

Standards for Graph Algorithm Primitives

Tim Mattson (Intel Corporation), David Bader (Georgia Institute of Technology), Jon Berry (Sandia National Laboratory), Aydin Buluc (Lawrence Berkeley National Laboratory), Jack Dongarra (University of Tennessee), Christos Faloutsos (Carnegie Mellon University), John Feo (Pacific Northwest National Laboratory), John Gilbert (University of California at Santa Barbara), Joseph Gonzalez (University of California at Berkeley), Bruce Hendrickson (Sandia National Laboratory), Jeremy Kepner (Massachusetts Institute of Technology), Charles Leiserson (Massachusetts Institute of Technology), Andrew Lumsdaine (Indiana University), David Padua (University of Illinois at Urbana-Champaign), Stephen Poole (Oak Ridge National Laboratory), Steve Reinhardt (Cray Corporation), Mike Stonebraker (Massachusetts Institute of Technology), Steve Wallach (Convey Corporation), Andrew Yoo (Lawrence Livermore National Laboratory)

Abstract-- It is our view that the state of the art in constructing a large collection of graph algorithms in terms of linear algebraic operations is mature enough to support the emergence of a standard set of primitive building blocks. This paper is a position paper defining the problem and announcing our intention to launch an open effort to define this standard.

http://graphblas.org
The case for sparse matrices

Many irregular applications contain coarse-grained parallelism that can be exploited by abstractions at the proper level.

<table>
<thead>
<tr>
<th>Traditional graph computations</th>
<th>Graphs in the language of linear algebra</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data driven, unpredictable communication.</td>
<td>Fixed communication patterns</td>
</tr>
<tr>
<td>Irregular and unstructured, poor locality of reference</td>
<td>Operations on matrix blocks exploit memory hierarchy</td>
</tr>
<tr>
<td>Fine grained data accesses, dominated by latency</td>
<td>Coarse grained parallelism, bandwidth limited</td>
</tr>
</tbody>
</table>
Linear-algebraic primitives for graphs

Sparse matrix X sparse matrix

Sparse matrix X sparse vector

Element-wise operations

Sparse matrix indexing

Is **think-like-a-vertex** really more productive?

“Our mission is to build up a linear algebra sense to the extent that vector-level thinking becomes as natural as scalar-level thinking.”

- Charles Van Loan
Examples of semirings in graph algorithms

<table>
<thead>
<tr>
<th>Semiring</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>((\mathbb{R}, +, \times))</td>
<td>Classical numerical linear algebra</td>
</tr>
<tr>
<td>(({0, 1},</td>
<td>, &amp;))</td>
</tr>
<tr>
<td>((\mathbb{R} \cup {\infty}, \min, +))</td>
<td>Shortest paths</td>
</tr>
<tr>
<td>((S, \text{select}, \text{select}))</td>
<td>Select subgraph, or contract nodes to form quotient graph</td>
</tr>
<tr>
<td>((\text{edge/vertex attributes, vertex data aggregation, edge data processing}))</td>
<td>Schema for user-specified computation at vertices and edges</td>
</tr>
<tr>
<td>((\mathbb{R}, \max, +))</td>
<td>Graph matching &amp; network alignment</td>
</tr>
<tr>
<td>((\mathbb{R}, \min, \times))</td>
<td>Maximal independent set</td>
</tr>
</tbody>
</table>

- **Shortened semiring notation:** \((\text{Set, Add, Multiply})\). Both identities omitted.
- **Add:** Traverses edges, **Multiply:** Combines edges/paths at a vertex
- Neither add nor multiply needs to have an inverse.
- Both add and multiply are associative, multiply distributes over add
Breadth-first search in the language of matrices

from

\[ A^T \]

to
Particular semiring operations:

**Multiply:** select

**Add:** minimum

Parents:

\[
\begin{bmatrix}
A^T & X \\
\end{bmatrix}
\]
Select vertex with minimum label as parent

from

to

parents:
• **Goal:** A crucial piece of the GraphBLAS effort is to translate the mathematical specification to an actual Application Programming Interface (API) that
  i. is faithful to the mathematics as much as possible, and
  ii. enables efficient implementations on modern hardware.

• **Impact:** All graph and machine learning algorithms that can be expressed in the language of linear algebra

• **Innovation:** Function signatures (e.g. mxm, vxm, assign, extract), parallelism constructs (blocking v. non-blocking), fundamental objects (masks, matrices, vectors, descriptors), a hierarchy of algebras (functions, monoids, and semiring)

```c
GrB_info GrB_mxm(GrB_Matrix *C, // destination
                 const GrB_Matrix Mask,
                 const GrB_BinaryOp accum,
                 const GrB_Semiring op,
                 const GrB_Matrix A,
                 const GrB_Matrix B
                 [, const Descriptor desc]);
```

\[ C(-M) \oplus = A^T \oplus \otimes B^T \]

What does Mask accomplish?

- Masks avoid computation and materialization of intermediate objects.
- All masks are “write” masks (i.e. they apply to the output)
- Any object (not just Boolean) can be passed as a mask
- The *structural complement* of a mask can be used *without materialization*
- Check the spec for the intricate semantics of mixing masks & accumulators.
- Masks are useful in domains outside graph analysis:
  - **Neural network** pruning: “Caffe was modified to add a mask which disregards pruned parameters during network operation for each weight tensor” (Han, Pool, Tran, and Dally, NIPS 2015)
  - Personalized PageRank (avoid converged vertices to receive messages)
  - “The sampled dense-dense matrix product (SDDMM) is written $P = A \ast_s B = (AB)$ \( (S > 0) \). $P$’s values are the elements of the product $AB$ evaluated at the nonzeros of $S$, and zero elsewhere. SDDMM is a bottleneck operation in all of the *factor analysis algorithms* (ALS, SFA, LDA, GaP).” (Canny and Zhao, 2013)
BFS in GraphBLAS with Masks

GrB_Vector q;
GrB_Vector_new(&q, GrB_BOOL, n);  // Vector<bool> q(n) = false
GrB_Vector_setElement(q, (bool)true, s);  // q[s] = true, false everywhere else

GrB_Monoid Lor;
GrB_Monoid_new(&Lor, GrB_LOR, false);  // Logical-or monoid

GrB_Semiring Boolean;
GrB_Semiring_new(&Boolean, Lor, GrB_LAND);  // Boolean semiring

GrB_Descriptor desc;  // Descriptor for vxm
GrB_Descriptor_new(&desc);
GrB_Descriptor_set(desc, GrB_MASK, GrB_SCMP);  // invert the mask
GrB_Descriptor_set(desc, GrB_OUTP, GrB_REPLACE);  // clear the output before assignment

GrB_UnaryOp apply_level;
GrB_UnaryOp_new(&apply_level, return_level, GrB_INT32, GrB_BOOL);

/*/  
  * BFS traversal and label the vertices.
  */
level = 0;
GrB_Index nvals;
do {
    ++level;  // next level (start with 1)
    GrB_apply(*v, GrB_NULL, GrB_PLUS_INT32, apply_level, q, GrB_NULL);  // v[q] = level
    GrB_vxm(q, *v, GrB_NULL, Boolean, q, A, desc);  // q[!v] = q || .&& A; finds all the
    GrB_Vector_nvals(&nvals, q);  // unvisited successors from current q
} while (nvals);  // if there is no successor in q, we are done.
Graph Algorithms on GraphBLAS

- **Miscellaneous:**
  - connectivity, traversal (BFS), independent sets (MIS), graph matching

- **Centrality**
  - (PageRank, betweenness, closeness)

- **Graph clustering**
  - (Markov cluster, peer pressure, spectral, local)

- **Shortest paths**
  - (all-pairs, single-source, temporal)

---

GraphBLAS primitives in increasing arithmetic intensity

- Sparse Matrix-Sparse Vector (SpMSpv)
- Sparse Matrix-Dense Vector (SpMV)
- Sparse Matrix Times Multiple Dense Vectors (SpMM)
- Sparse - Sparse Matrix Product (SpGEMM)
- Sparse - Dense Matrix Product (SpDM³)
A work-efficient parallel algorithm for sparse matrix-sparse vector multiplication (SpMSpV)

• **Goal:** A scalable SpMSpV algorithm without doing more work on higher concurrency

• **Application:** Breadth-first search, graph matching, support vector machines, etc.

• **Algorithmic innovation:**
  - Attains work-efficiency by arranging necessary columns of the matrix into buckets where each bucket is processed by a single thread
  - Avoids synchronization by row-wise partitioning of the matrix on the fly

• **Performance:**
  - First ever work-efficient algorithm for SpMSpV that attains up to 15x speedup on a 24-core Intel Ivy Bridge processor and up to 49x speedup on a 64-core KNL processor
  - Up to an order of magnitude faster than its competitors, especially for sparser vector

---

A.Azad, A. Buluç. A work-efficient parallel sparse matrix-sparse vector multiplication algorithm. IPDPS’17
Push-pull $\equiv$ column-row matvec!

Yang, C., Buluc, A. and Owens, J.D., Implementing Push-Pull Efficiently in GraphBLAS. ICPP ’18
Optimization 1: change of direction

- Generalizes to all algorithms that use sparse matvec (SpMSpV).
- Previously known and used with success in Ligra.
Optimization 2: Masking

Row-based matvec w/ mask

Column-based matvec w/ mask

Complexity: $O(dN) \rightarrow O(d \, nnz(m))$
- $d$: average vertex degree
- $nnz(m)$ is the number nonzeros in the mask
- $N$ is the matrix/vector length
Optimization 3: Early-exit

- Only works for Boolean semirings as it exploits short-circuitting.
Machine Learning for Science

Classification

Classification + Localization

Object Detection

Instance Segmentation

Slide source: Prabhat
Machine Learning relies a lot on Linear Algebra

Higher-level machine learning tasks

Logistic Regression, Support Vector Machines
Dimensionality Reduction (NMF, CX, PCA)
Clustering (e.g., MCL, Spectral Clustering)
Partial Correlation Estimation (CONCORD)
Deep Learning (Neural Nets)

Graph/Sparse/Dense BLAS functions (in increasing arithmetic intensity)
Implicit Parallelization: Keep the overall algorithm structure (the sequence of operations) intact and parallelize the individual operations.

Example: parallelizing the BLAS operations in previous figure
+ Often achieves exactly the same accuracy (e.g., model parallelism in DNN training)
- Scalability can be limited if the critical path of the algorithm is long

Explicit Parallelization: Modify the algorithm to extract more parallelism, such as working on individual pieces whose results can later be combined

Examples: CA-SVM and data parallelism in DNNs
+ Significantly better scalability can be achieved
- No longer the same algorithmic properties (e.g. HogWild!).
Philosophy of the Markov Cluster Algorithm (MCL)

The number of **edges or higher-length paths** between two arbitrary nodes in a cluster is greater than the number of paths between nodes from different clusters.

**Random walks** on the graph will frequently remain within a cluster.

The algorithm computes the probability of random walks through the graph and removes lower probability terms to form clusters.
Markov Cluster Algorithm (MCL)

Widely popular and successful algorithm for discovering clusters in protein interaction and protein similarity networks

At each iteration:

**Step 1 (Expansion):** Squaring the matrix while pruning (a) small entries, (b) denser columns

**Naïve implementation:** sparse matrix-matrix product (SpGEMM), followed by column-wise top-K selection and column-wise pruning

**Step 2 (Inflation):** taking powers entry-wise
A combined expansion and pruning step

- **b**: number of columns in the output constructed at once
  - Smaller b: less parallelism, memory efficient (b=1 is equivalent to sparse matrix-sparse vector multiplication used in MCL)
  - Larger b: more parallelism, memory intensive
A combined expansion and pruning step

- $b$: number of columns in the output constructed at once
  - HipMCL selects $b$ dynamically as permitted by the available memory
  - The algorithm works in $h = \frac{N}{b}$ phases where $N$ is the number of columns (vertices in the network) in the matrix

$$A \times A^2 = \text{Prune}(A^2)$$
HipMCL: High-performance MCL

- MCL process is both **computationally expensive** and **memory hungry**, limiting the sizes of networks that can be clustered.
- HipMCL overcomes such limitation via **sparse parallel algorithms**.
- **Up to 1000X times faster** than original MCL with same accuracy.

### HipMCL on large networks

<table>
<thead>
<tr>
<th>Data</th>
<th>Proteins</th>
<th>Edges</th>
<th>#Clusters</th>
<th>HipMCL time</th>
<th>platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>Isolate-1</td>
<td>47M</td>
<td>7 B</td>
<td>1.6M</td>
<td>1 hr</td>
<td>1024 nodes Edison</td>
</tr>
<tr>
<td>Isolate-2</td>
<td>69M</td>
<td>12 B</td>
<td>3.4M</td>
<td>1.66 hr</td>
<td>1024 nodes Edison</td>
</tr>
<tr>
<td>Isolate-3</td>
<td>70M</td>
<td>68 B</td>
<td>2.9M</td>
<td>2.41 hr</td>
<td>2048 nodes Cori KNL</td>
</tr>
<tr>
<td>MetaClust50</td>
<td>282M</td>
<td>37B</td>
<td>41.5M</td>
<td>3.23 hr</td>
<td>2048 nodes Cori KNL</td>
</tr>
</tbody>
</table>

MCL can not cluster these networks
The computation cube of matrix-matrix multiplication

Matrix multiplication: \( \forall (i,j) \in n \times n, \quad C(i,j) = \sum_k A(i,k)B(k,j), \)

The computation (discrete) cube:
- A face for each (input/output) matrix
- A grid point for each multiplication

How about sparse algorithms?
3D parallel SpGEMM in a nutshell

\[ C_{int}^{ijk} = \sum_{l=1}^{p/c} A_{ilk} B_{ljk} \]

3D SpGEMM performance

2D (non-threaded) is the previous state-of-the-art.

3D (threaded) – first presented here – beats it by 8X at large concurrencies.

Strong scaling of different variants of 2D and 3D algorithms when squaring of nlpkkt160 matrix on Edison.
Sparse Inverse Covariance Matrix Estimation

- **Precision matrix = Inverse covariance matrix**
- **Goal:** Estimating graphical model structure
- “The zeros of a precision matrix correspond to zero partial correlation, a necessary and sufficient condition for conditional independence (Lauritzen, 1996)”
- Sparsity often enforced by regularization
- One algorithm (HP-CONCORD)’s objective function:
  \[
  \min_{\Omega \in \mathbb{R}^{p \times p}} -\log \det(\Omega^2_D) + \text{tr}(\Omega S\Omega) + \lambda_1 \|\Omega_X\|_1 + \frac{\lambda_2}{2} \|\Omega\|_F^2,
  \]
- \(\Omega\) is the sparse inverse covariance matrix we are trying to estimate
Why care? Finding Direct Associations

Partial Correlation (a.k.a. sparse inverse covariance estimation): direct association without confounders

- Gene Regulatory Network (GRN) estimation
- Joint modeling of SNPs and GRN
- Linkage Disequilibrium (LD) estimation
- Canonical Correlation Analysis (CCA)
- Genome-wide association studies (GWAS)

Data-driven hypothesis generation!

- Computationally challenging;
- HP-CONCORD on distributed memory increases scalability
HP-CONCORD Advantages

- HP-CONCORD makes fewer assumptions about the data (in particular, no Gaussianity is assumed) compared to competitors
- Thanks to **communication-avoiding matrix algorithms**, it reaches unprecedented scales
  - BigQUIC: previous state-of-the-art
  - Obs-K are our HP-CONCORD algorithm (K: number of nodes)
  - Experiment is trying to recover a random graph structure.

---

The best algorithm when multiplying two matrices with unequal nonzero counts?

- Depends on the concurrency!
Training Neural Networks

- Training is to **adjust the weights** (\(W\)) in the connections of the neural network, in order to change the function it represents.

\[
\begin{align*}
  w_{1,3} & \quad w_{1,4} & \quad w_{1,5} & \quad w_{2,3} & \quad w_{2,4} & \quad w_{2,5} \\
  w_{3,6} & \quad w_{4,6} & \quad w_{5,6} \\
\end{align*}
\]

A “shallow” neural network with only one hidden layer (nodes 3,4,5), two inputs and one output.

\(W\): the matrix of weights

**Diagram:**
- **Nodes:** 1, 2, 3, 4, 5, 6
- **Connections:**
  - \(w_{1,3}\) from 1 to 3
  - \(w_{1,4}\) from 1 to 4
  - \(w_{1,5}\) from 1 to 5
  - \(w_{2,3}\) from 2 to 3
  - \(w_{2,4}\) from 2 to 4
  - \(w_{2,5}\) from 2 to 5
  - \(w_{3,6}\) from 3 to 6
  - \(w_{4,6}\) from 4 to 6
  - \(w_{5,6}\) from 5 to 6

**Note:** Only parameters are weights for simplicity (i.e. ignore bias parameters).
Gradient Descent

\[ W^{t+1} \leftarrow W^t - \alpha \cdot \nabla_W f(W^t, x) \]

- Also called the steepest descent algorithm
- In order to minimize a function, move towards the opposite direction of the gradient at a rate of \( \alpha \).
- \( \alpha \) is the step size (also called the learning rate)
- Used as the **optimization backend** of many other machine learning methods (example: NMF)
SGD training of NNs as matrix operations

\[ \mathbf{W}: \text{weights} \]

\[ \mathbf{X} = \] \[ \mathbf{B} \]

\[ \mathbf{X}_{\text{out}}: \text{outputs of this layer} \]

\[ \mathbf{X}_{\text{in}}: \text{inputs to this layer} \]

\( N = \) the number of outputs
\( M = \) the number of inputs
\( B = \) the size of the minibatch

**The impact to parallelism:**

- \( \mathbf{W} \) is replicated to processor, so it doesn’t change
- \( \mathbf{X}_{\text{in}} \) and \( \mathbf{X}_{\text{out}} \) gets skinnier if we only use data parallelism, i.e. distributing \( b = \frac{B}{p} \) mini-batches per processor
- GEMM performance suffers as *matrix dimensions get smaller* and *more skewed*
- **Result:** Data parallelism can hurt single-node performance
Data Parallel SGD training of NNs as matrix operations

1. Which matrices are replicated?
2. Where is the communication?
3. Which steps can be overlapped?

\[ \nabla_Y = \frac{\partial L}{\partial Y} = \text{how did the loss function change as output activations changed?} \\
\nabla_X = \frac{\partial L}{\partial X} \\
\nabla_W = \frac{\partial L}{\partial W} \]
Model Parallel SGD training of NNs as matrix operations

1. Which matrices are replicated?
2. Where is the communication?
3. How can matrix algebra capture both model and data parallelism?
Data & Model Parallel SGD training of NNs as matrix operations

Processes are 2D indexed: \( P = P_r \times P_c \)

Conclusions

- Both graph algorithms and machine learning have growing importance in scientific applications.
- Not everything is [sparse] linear algebra, but a lot of things are!
- Transfer of techniques and knowledge is easier when your scientific base is not domain specific.
- Communication-avoiding [sparse] linear algebra algorithms provide unprecedented scaling for problems outside traditional scientific computing, such as computational biology, graph analysis, and machine learning.
- Check out http://graphblas.org, HipMCL, and HP-CONCORD.
Acknowledgments


My work is funded by U.S. Department of Energy Office of Science.