Gilbert: Declarative Sparse Linear Algebra on Massively Parallel Dataflow Systems

Till Rohrmann ¹  Sebastian Schelter ²  Tilmann Rabl ²  Volker Markl ²

¹Apache Software Foundation
²Technische Universität Berlin

March 8, 2017
Motivation
Information Age

- Collected data grows exponentially
- Valuable information stored in data
- *Need for scalable analytical methods*
Distributed Computing and Data Analytics

- Writing parallel algorithms is tedious and error-prone
- Huge existing code base in form of libraries
- Need for parallelization tool
Requirements

- Linear algebra is lingua franca of analytics
- Parallelize programs automatically to simplify development
- Sparse operations to support sparse problems efficiently

Goal

Development of distributed sparse linear algebra system
Gilbert
**Gilbert in a Nutshell**

**Linear Algebra Program**

```matlab
eps = 0.0001;
maxIterations = 20;
d = sum(A,2);
% create the column-stochastic transition matrix
T = (diag(1 ./ d) * A)';

% initialize the ranks
r = ones(numVertices, 1) / numVertices;

% compute PageRank
e = ones(numVertices, 1)/numVertices;

for i = 1:maxIterations
    rNew = .85 * T * r + .15 * e;
    value = norm(rNew-r);
    r = rNew;
    if(value < eps)
        break
    end
end
```

**Distributed Execution**
System architecture

- Language Layer
  - Lexer & Parser
  - Typer
  - Compiler

- Intermediate Layer
  - Optimizer
  - Execution Plan Generator

- Runtime Layer
  - Local
  - Flink
  - Spark
Gilbert Language

- Subset of MATLAB® language
- Support of basic linear algebra operations
- Fixpoint operator serves as side-effect free loop abstraction
- Expressive enough to implement a wide variety of machine learning algorithms

1. \( A = \text{rand}(10, 2); \)
2. \( B = \text{eye}(10); \)
3. \( A' * B; \)
4. \( f = @(x) x.^2.0; \)
5. \( \text{eps} = 0.1; \)
6. \( c = @(p, c) \text{norm}(p-c, 2) < \text{eps}; \)
7. \( \text{fixpoint}(1/2, f, 10, c); \)
Gilbert Typer

- Matlab is dynamically typed
- Dataflow systems require type knowledge at compile time
- Automatic type inference using the Hindley-Milner type inference algorithm
- Infer also matrix dimensions for optimizations

```
1  A = rand(10, 2): Matrix(Double, 10, 2)
2  B = eye(10): Matrix(Double, 10, 10)
3  A' * B: Matrix(Double, 2, 10)
4  f = @(x) x.^2.0: N -> N
5  eps = 0.1: Double
6  c = @(p,c) norm(p-c,2) < eps: (N,N) -> Boolean
7  fixpoint(1/2, f, 10, c): Double
```
Intermediate Representation & Gilbert Optimizer

- Language independent representation of linear algebra programs
- Abstraction layer facilitates easy extension with new programming languages (such as R)
- Enables language independent optimizations
  - Transpose push down
  - Matrix multiplication re-ordering
Distributed Matrices

(a) Row partitioning

(b) Quadratic block partitioning

Which partitioning is better suited for matrix multiplications?

\[ \text{io\_cost}_{\text{row}} = \mathcal{O}(n^3) \]
\[ \text{io\_cost}_{\text{block}} = \mathcal{O}(n^2 \sqrt{n}) \]
Apache Flink and Apache Spark offer MapReduce-like API with additional operators: \texttt{join}, \texttt{coGroup}, \texttt{cross}
Evaluation
Gaussian Non-Negative Matrix Factorization

- Given $V \in \mathbb{R}^{d \times w}$ find $W \in \mathbb{R}^{d \times t}$ and $H \in \mathbb{R}^{t \times w}$ such that $V \approx WH$
- Used in many fields: Computer vision, document clustering and topic modeling
- Efficient distributed implementation for MapReduce systems

Algorithm

$$H \leftarrow \text{randomMatrix}(t, w)$$
$$W \leftarrow \text{randomMatrix}(d, t)$$

while $\|V - WH\|_2 > \text{eps}$ do
  $$H \leftarrow H \cdot (W^T V / W^T WH)$$
  $$W \leftarrow W \cdot (VH^T / WHH^T)$$
end while
Testing Setup

- Set $t = 10$ and $w = 100000$
- $V \in \mathbb{R}^{d \times 100000}$ with sparsity $0.001$
- Block size $500 \times 500$
- Numbers of cores 64
- Flink 1.1.2 & Spark 2.0.0
- Gilbert implementation: 5 lines
- Distributed GNMF on Flink: 70 lines

```
1 V = rand($rows, 100000, 0, 1, 0.001);
2 H = rand(10, 100000, 0, 1);
3 W = rand($rows, 10, 0, 1);
4 nH = H.((W' * V) ./ (W' * W * H))
5 nW = W.((V * nH') ./ (W * nH * nH'))
```
**Gilbert Optimizations**

![Graph showing execution time vs rows](image)

- Optimized Spark
- Optimized Flink
- Non-optimized Spark
- Non-optimized Flink

**Execution time** $t$ in s vs **Rows $d$ of $V$**
Optimizations Explained

Matrix updates

\[ H \leftarrow H \cdot \left( W^T V / W^T WH \right) \]
\[ W \leftarrow W \cdot \left( VH^T / WHH^T \right) \]

Non-optimized matrix multiplications

\[ \in \mathbb{R}^{10 \times 100000} \]  
\[ \left( W^T W \right) H \]  
\[ \in \mathbb{R}^{10 \times 10} \]  
\[ \left( WH \right) H^T \]  
\[ \in \mathbb{R}^{d \times 10} \]  
\[ \mathbb{R}^{d \times 100000} \]

Optimized matrix multiplications

\[ \in \mathbb{R}^{10 \times 100000} \]  
\[ \left( W^T W \right) H \]  
\[ \in \mathbb{R}^{10 \times 10} \]  
\[ W \left( HH^T \right) \]  
\[ \in \mathbb{R}^{10 \times 10} \]

\[ \mathbb{R}^{d \times 10} \]  
\[ \in \mathbb{R}^{10 \times 100000} \]
GNMF Step: Scaling Problem Size

- Distributed Gilbert execution handles much larger problem sizes than local execution
- Specialized implementation is slightly faster than Gilbert
Both distributed backends show good weak scaling behaviour
PageRank

- Ranking between entities with reciprocal quotations and references

\[ PR(p_i) = d \sum_{p_j \in L(p_i)} \frac{PR(p_j)}{D(p_j)} + \frac{1 - d}{N} \]

- \( N \) - number of pages
- \( d \) - damping factor
- \( L(p_i) \) - set of pages being linked by \( p_i \)
- \( D(p_i) \) - number of linked pages by \( p_i \)
- \( M \) - transition matrix derived from adjacency matrix

\[ R = d \cdot MR + \frac{1 - d}{N} \cdot \mathbb{1} \]
PageRank Implementation

**MATLAB®**

1. `it = 10;`
2. `d = sum(A, 2);`
3. `M = (diag(1 ./ d) * A)’;`
4. `r_0 = ones(n, 1) / n;`
5. `e = ones(n, 1) / n;`
6. `for i = 1:it`
7. `r = .85 * M * r + .15 * e`
8. `end`

**Gilbert**

1. `it = 10;`
2. `d = sum(A, 2);`
3. `M = (diag(1 ./ d) * A)’;`
4. `r_0 = ones(n, 1) / n;`
5. `e = ones(n, 1) / n;`
6. `fixpoint(r_0, @r) .85 * M * r + .15 * e, it) `
PageRank: 10 Iterations

- Gilbert backends show similar performance
- Specialized implementation faster because it can fuse operations
Conclusion
Conclusion

- Easy to use sparse linear algebra environment for people familiar with MATLAB®
- Scales to data sizes exceeding a single computer
- High-level linear algebra optimizations improve runtime
- Slower than specialized implementations due to abstraction overhead