SPOOF: Sum-Product Optimization and Operator Fusion for Large-Scale Machine Learning

Tarek Elgamal\textsuperscript{2}, Shangyu Luo\textsuperscript{3}, Matthias Boehm\textsuperscript{1}, Alexandre V. Evfimievski\textsuperscript{1}, Shirish Tatikonda\textsuperscript{4}, Berthold Reinwald\textsuperscript{1}, Prithviraj Sen\textsuperscript{1}

\textsuperscript{1} IBM Research – Almaden
\textsuperscript{3} Rice University
\textsuperscript{2} University of Illinois
\textsuperscript{4} Target Corporation
Motivation

- **Declarative Large-Scale Machine Learning (ML)**
  - Simplify development / usage of ML tasks or algorithms
  - SystemML: High-level language → data independence / plan generation
  - State-of-the-art compilers: rewrites, operator selection, fused operators

- **Ubiquitous Optimization Opportunities**
  - Example Rewrites: \( X^T y \rightarrow (y^T X)^T \), \( \text{sum}(\lambda X) \rightarrow \lambda \text{sum}(X) \), \( \text{trace}(X Y) \rightarrow \text{sum}(X \odot Y^T) \)
  - Example Fused operators: \( \text{sum}(X \odot Y \odot Z) \), \( X^T(X v) \), \( \text{sum}(X \odot \log(U V^T)) \)
Motivation

- **Declarative Large-Scale Machine Learning (ML)**
  - Simplify development / usage of ML tasks or algorithms
  - SystemML: High-level language → data independence / plan generation
  - State-of-the-art compilers: rewrites, operator selection, fused operators

- **Ubiquitous Optimization Opportunities**
  - Example Rewrites: \(X^T y \rightarrow (y^T X)^T, \text{sum}(\lambda X) \rightarrow \lambda \text{sum}(X),\)
    \(\text{trace}(X Y) \rightarrow \text{sum}(X \odot Y^T)\)
  - Example Fused operators: \(\text{sum}(X \odot Y \odot Z), X^T(X v), \text{sum}(X \odot \log(U V^T))\)
  - Fewer intermediates, fewer scans, sparsity exploitation, less compute

- **Problems and Challenges**
  - Large Development Effort: number of patterns, multiple runtime back-ends, multiple formats and combinations (sparse/dense)
  - High Performance Impact: slightly changed patterns can render rewrites and fused operators inapplicable
Example PNMF – A 1000x War Story

- Poisson Nonnegative Matrix Factorization (PNMF)
  - \( X \approx WH \) of low rank \( k \); \( X: 200K \times 200K, sp=0.001 \) (480MB)

```
1: X = read("./input/X")
2: k = 100; eps = 1e-15; max_iter = 10; iter = 1;
3: W = rand(rows=nrow(X), cols=k, min=0, max=0.025)
4: H = rand(rows=k, cols=ncol(X), min=0, max=0.025)
5: while( iter < max_iter ) {
   6:   H = (H*(t(W)*((X/(W*H+eps))))) / t(colSums(W));
   7:   W = (W*((X/(W*H+eps))*t(H))) / t(rowSums(H));
   8:   obj = sum(W*H) - sum(X*log(W*H+eps));
   9:   print("iter=", iter + " obj=", obj);
  10:  iter = iter + 1;
11: }
12: write(W, "./output/W");
13: write(H, "./output/H");
```

The problem is \( WH \) (320GB), but we could add sparsity-exploiting fused operators and rewrites

\( \rightarrow \) rewrites and fused operators: 1000x
Our Vision: Holistic Optimization Framework

- **SPOOF Compiler Framework**
  - Automatic rewrite identification and operator fusion
  - Increased opportunities and side effects (CSE, rewrites $\leftrightarrow$ fusion)
  - **Key ideas:** (1) break up LA operations into basic operators (in RA),
    (2) elementary sum-product and RA rewrites, and (3) fused operator generation

\[
\text{sum}(W H) \left( \frac{X}{(W H + \epsilon)} \right) H^T
\]

\[
\text{colSums}(W) \ast \text{rowSums}(H) \quad \text{Op TMP5}
\]
Sum-Product Optimization

- **SP-Plan Representation: restricted relational algebra**
  - **Data**: input matrices are relations of \((i, j, v)\)-tuples (intermediates are tensors)
  - **Basic operations**: selection \(\sigma\), extended projection \(\Pi\), aggregation \(\Gamma\), join \(\bowtie\)
  - **Composite operations**: e.g., multiply \(A_{ij} *, i=k \land j=l \ B_{kl} := \Pi_{i,j;a+b}(A_{ija} \bowtie_{i=k \land j=l} B_{klb})\)
  - **Two restrictions**: a single value attribute per relation, and unique composite indexes per relation ➔ **single value per tensor cell**

- **Example SP Plan**
  - \(\text{sum}(W \ H)\)
Sum-Product Optimization, cont.

- Example SP Plan Rewrites
  - $W := 200K \times 100$, $H := 100 \times 200K$

But, SP opt alone can be counter-productive (e.g., CSE ‘W H’)

Aggregation Elimination

Sum-Product (distributive law)

Sum-Product (distributive law)

8.04 TFLOPs

8 TFLOPs

60 MFLOPs

40 MFLOPs
Operator Fusion

- **C-Plan Representation**
  - **Hybrid approach**: hand-coded operator skeletons with custom body code
    - Efficiency (data access, multi-threading) and flexibility
  - **Template C-Nodes**: generic fused operator skeletons (w/ data binding)
    e.g., SpoofOuterProduct, SpoofCellwise, SpoofRowAggregate
  - **Primitive C-Nodes**: vector/scalar operations

- **Example C-Plan**
  - \((X / (W H + \epsilon)) H^T\)
    (PNMF update rule)
Operator Fusion, cont.

- **Example C-Plan Codegen**
  - Recursive codegen on C-Plan
  - **Generated operator inherits data access, multi-threading, etc from template skeleton**

```java
1: public final class TMP5 extends SpoofOuterProduct {
2:     public TMP5() {
3:         _type = OuterProductType.RIGHT;
4:     }
5:     protected void exec(double a, double[] b, int bi,
6:                             double[] c, int ci,..., double[] d, int di, int k)
7:         {
8:             double TMP1 = dotProduct(b, c, bi, ci, k);  // WH
9:             double TMP2 = TMP1 + 1.0E-15;        // +eps
10:            double TMP3 = a / TMP2;                // X/
11:            vectMultiplyAdd(TMP3, c, d, ci, di, k); // t(H)
12:         }
13: }
```
Experimental Setting

- **Cluster Setup**
  - 1 head node (2x4 Intel E5530, 64GB RAM), and
    6 worker nodes (2x6 Intel E5-2440, 96GB RAM, 12x2TB disks)
  - Spark 1.5.2 with 6 executors (24 cores, 60GB), 30GB driver memory

- **ML Programs and Data**
  - 3 full-fledged ML algorithms (PNMF, L2SVM, Mlogreg)
  - Synthetically generated data

- **Selected Baselines**
  - Apache SystemML 0.10 (May 2016): Base, Fused, SPOOF
  - Julia 0.5 (Sep 2016) w/ LLVM-based just-in-time compiler
Micro Benchmarks: Operations Performance
(@ single worker node)

\[(X / (WH + \varepsilon))H^T, \text{ (PNMF)}\]
10K x 10K, k=100, Multi-threaded

\[\text{sum}(X \odot Y \odot Z), \text{ (L2SVM)}\]
dense, Multi-threaded

➔ Sparsity-exploiting operators at 1/12 peak compute bandwidth

➔ Fused operator w/o intermediates at peak 1xlocal / remote memory bandwidth (25GB/s)
End-to-End Experiments: PNMF and LSVM

**PNMF Execution Time** (incl. compilation and I/O)
- 20 iterations, rank $k = 100$

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Base</th>
<th>Fused</th>
<th>SPOOF</th>
</tr>
</thead>
<tbody>
<tr>
<td>10K x 10K, 0.001</td>
<td>251 s</td>
<td>6 s</td>
<td>9 s</td>
</tr>
<tr>
<td>25K x 25 K, 0.001</td>
<td>4,748 s</td>
<td>9 s</td>
<td>11 s</td>
</tr>
<tr>
<td>200K x 200K, 0.001</td>
<td>&gt;24h</td>
<td>121 s</td>
<td>125 s</td>
</tr>
</tbody>
</table>

**L2SVM Execution Time** (incl. compilation and I/O)
- 20 outer iterations, $\epsilon = 10^{-14}$

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Base</th>
<th>Fused</th>
<th>SPOOF</th>
</tr>
</thead>
<tbody>
<tr>
<td>100K x 10, 1.0 (8MB)</td>
<td>3 s</td>
<td>3 s</td>
<td>5 s</td>
</tr>
<tr>
<td>1M x 10, 1.0 (80MB)</td>
<td>9 s</td>
<td>7 s</td>
<td>8 s</td>
</tr>
<tr>
<td>10M x 10, 1.0 (800MB)</td>
<td>50 s</td>
<td>34 s</td>
<td>17 s</td>
</tr>
<tr>
<td>100M x 10, 1.0 (8GB)</td>
<td>525 s</td>
<td>320 s</td>
<td>114 s</td>
</tr>
</tbody>
</table>
Conclusions

- **Summary**
  - **SPOOF:** Automatic rewrite identification and operator fusion
  - Non-invasive compiler/runtime integration into SystemML
  - Plan representation/compilation for sum-product and codegen

- **Conclusions and Future Work**
  - Many rewrite/fusion opportunities with huge performance impact
  - Performance close to hand-coded ops w/ moderate compilation overhead
  - Future work: distributed operations, optimization algorithms

- **Available Open Source (soon)**
  - SYSTEMML-448: Code Generation, experimental in 1.0 release
  - Sum-product optimization and fusion optimizations later
SystemML is Open Source:
Apache Incubator Project since 11/2015
Website: http://systemml.apache.org/
Sources: https://github.com/apache/incubator-systemml
Backup: Operator Fusion, cont.

- **Example C-Plan Codegen**
  - L2SVM inner loop

```java
public final class TMP2 extends SpoofCellwise {
    public TMP2() {
        _type = CellType.FULL_AGG;
    }
    protected double exec(double a, double[][] vectors, double[] scalars,.., int rowIndex) {
        double TMP3 = vectors[1][rowIndex];
        double TMP4 = vectors[0][rowIndex];
        double TMP5 = a * scalars[0];
        double TMP6 = TMP4 + TMP5;
        double TMP7 = TMP3 * TMP6;
        double TMP8 = 1 - TMP7;
        double TMP9 = (TMP8 > 0) ? 1 : 0;
        double TMP10 = TMP8 * TMP9;
        double TMP11 = TMP10 * TMP3;
        double TMP12 = TMP11 * a;
        return TMP12;
    }
}
```

# Intermediates: Base: 10, Fused: 5, Spoof: 0

1. `out = 1 - Y * (Xw + step_sz*Xd);`
2. `sv = (out > 0);`
3. `out = out * sv;`
4. `g = wd+step_sz*dd - sum(out*Y*Xd);`
5. `h = dd + sum(Xd*sv*Xd);`
6. `step_sz = step_sz - g/h;`
Backup: Plan Caching Effects for Mlogreg

- **Dynamic Recompilation**
  - Problem of unknown or changing sizes (e.g., UDFs, data-dep. ops, size expr.)
  - Integration of Spoof into dynamic recompiler ➔ huge compilation overhead
  ➔ **Plan cache:** reuse compiled ops across DAGs / recompiations

- **Mlogreg Cache Statistics**
  - 500K x 200 (800MB), 20/5 outer/inner iterations, $\varepsilon = 10^{-14}$
  - CSLH: Context-sensitive literal heuristic

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Spoof no PC</th>
<th>Spoof constant PC</th>
<th>Spoof CSLH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution time</td>
<td>49.29 s</td>
<td>19.87 s</td>
<td>14.48 s</td>
</tr>
<tr>
<td>PC hit rates</td>
<td>0 / 462</td>
<td>388 / 462</td>
<td>449 / 462</td>
</tr>
<tr>
<td>Javac compile time (sync)</td>
<td>34.45 s</td>
<td>6.88 s</td>
<td>1.97 s</td>
</tr>
<tr>
<td>JIT compile time (async)</td>
<td>25.36 s</td>
<td>18.84 s</td>
<td>10.50 s</td>
</tr>
</tbody>
</table>