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

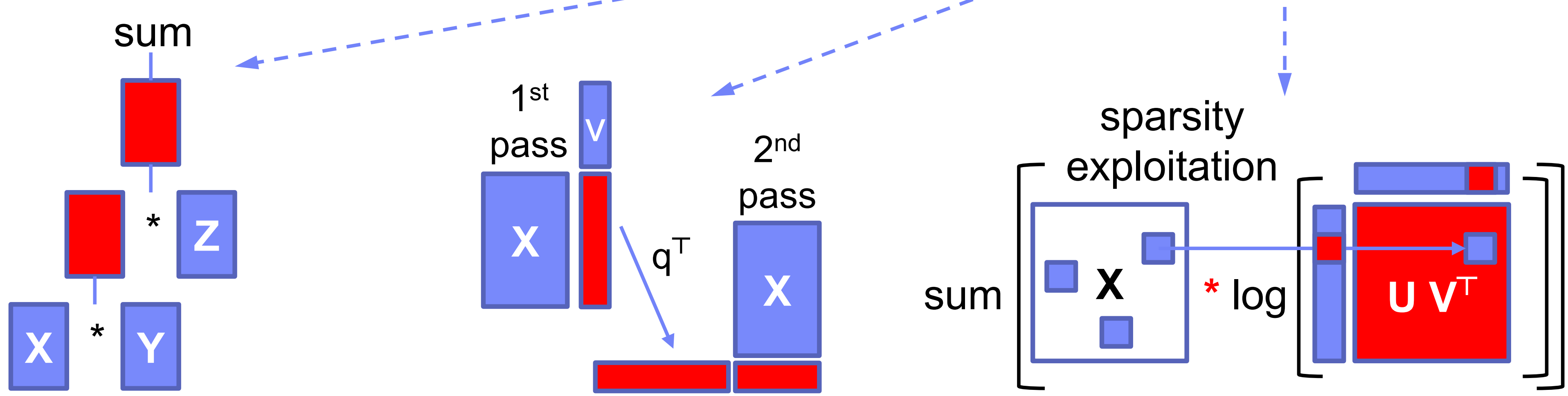
Tarek Elgamal[2], **Shangyu Luo**[3], **<u>Matthias Boehm</u>**[1], **Alexandre V. Evfimievski**[1], **Shirish Tatikonda**[4], **Berthold Reinwald**[1], **Prithviraj Sen**[1]

[1] IBM Research – Almaden
[2] University of Illinois
[3] Rice University
[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: $\mathbf{X}^\top \mathbf{y} \rightarrow (\mathbf{y}^\top \mathbf{X})^\top$, $\mathrm{sum}(\lambda\, \mathbf{X}) \rightarrow \lambda\, \mathrm{sum}(\mathbf{X})$, $\mathrm{trace}(\mathbf{X}\, \mathbf{Y}) \rightarrow \mathrm{sum}(\mathbf{X} \odot \mathbf{Y}^\top)$
  - Example Fused operators: $\mathrm{sum}(\mathbf{X} \odot \mathbf{Y} \odot \mathbf{Z})$, $\mathbf{X}^\top(\mathbf{X}\, \mathbf{v})$, $\mathrm{sum}(\mathbf{X} \odot \log(\mathbf{U}\, \mathbf{V}^\top))$



Motivation

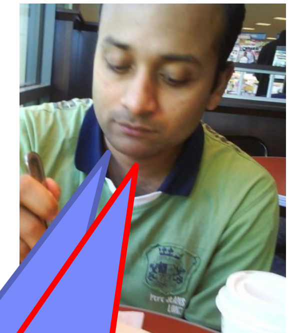
- **Declarative Large-Scale Machine Learning (ML)**
  - Simplify development / usage of ML tasks or algorithms
  - SystemML: High-level language → data independence / plan generation
  - State-of-the-art compilers: **rewrites, operator selection, fused operators**

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

  ➔ **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 ≈ W H** of low rank k; **X:** 200K x 200K, sp=0.001 (480MB)

```
X = read("./input/X")
k = 100; eps = 1e-15; max_iter = 10; iter = 1;
W = rand(rows=nrow(X), cols=k, min=0, max=0.025)
H = rand(rows=k, cols=ncol(X), min=0, max=0.025)
while( iter < max_iter ) {
   H = (H*(t(W)%*%(X/(W%*%H+eps)))) / t(colSums(W));
   W = (W*((X/(W%*%H+eps))%*%t(H))) / t(rowSums(H));
   obj = sum(W%*%H) - sum(X*log(W%*%H+eps));
   print("iter=" + iter + " obj=" + obj);
   iter = iter + 1;
}
write(W, "./output/W");
write(H, "./output/H");
```

It still takes forever – btw, I changed it slightly

Here is an interesting rewrite: sum(**W H**) → colSums(**W**) rowSums(**H**)

The problem is **W H** (320GB), but we could add sparsity-exploiting **fused operators and rewrites**

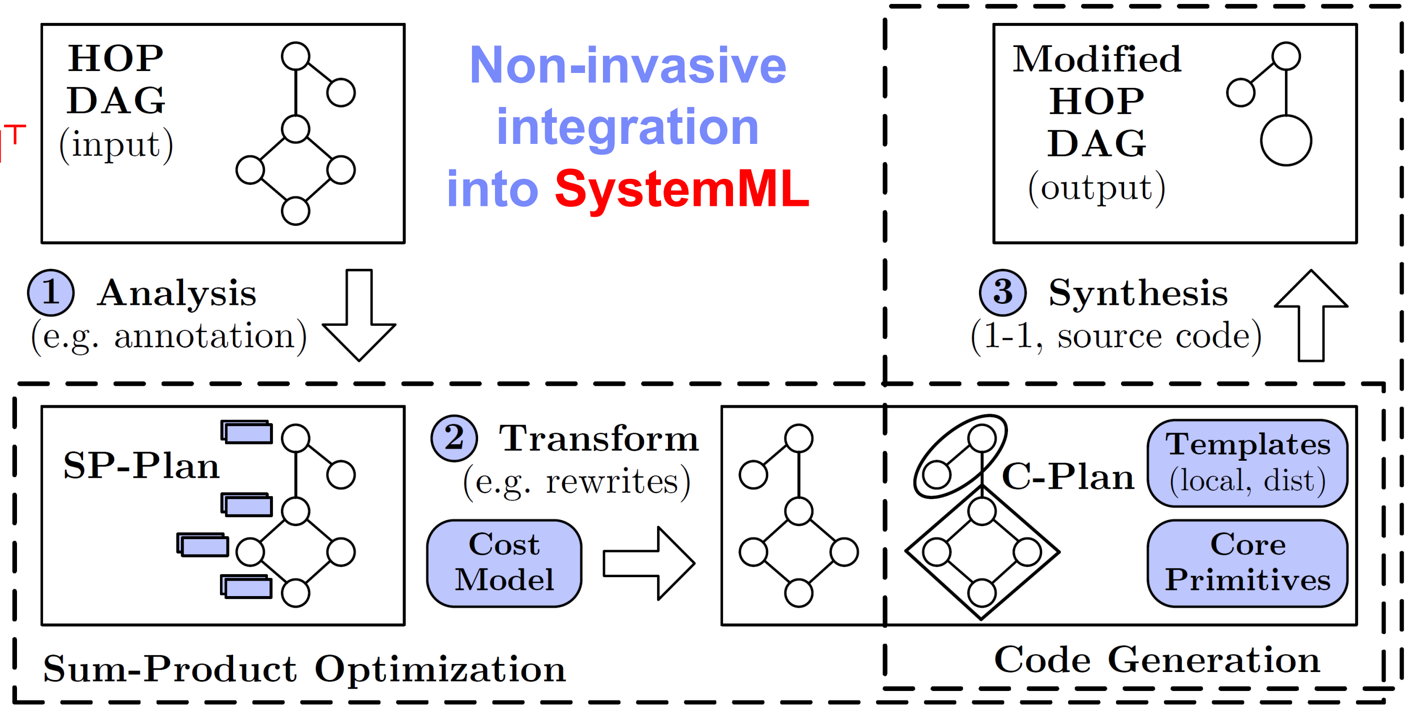
→ **rewrites and fused operators: 1000x**

Our Vision: Holistic Optimization Framework

- **SPOOF Compiler Framework**
  - **Automatic rewrite identification and operator fusion**
  - Increased opportunities and side effects (CSE, rewrites ←→ 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

sum(W H)

$(X/(W\,H + \varepsilon))H^T$



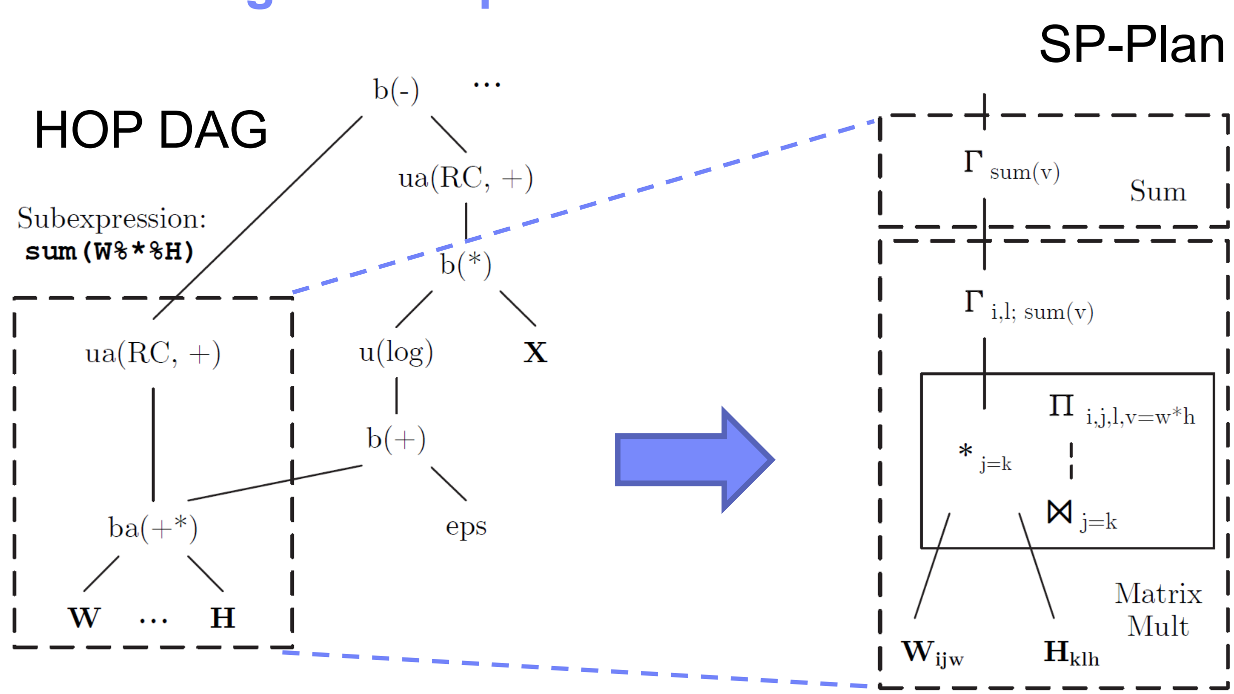
colSums(W) * rowSums(H)

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 σ, extended projection Π, aggregation Γ, join $\bowtie$
  - **Composite operations:** e.g., multiply $A_{ij} *_{i=k \wedge j=l} B_{kl} := \Pi_{i,j;a*b}(A_{ija} \bowtie_{i=k \wedge j=l} B_{klb})$
    addition $A_{ij} +_{i=k \wedge j=l} B_{kl} := \Pi_{i,j;a+b}(A_{ija} \text{⟗}_{i=k \wedge j=l} B_{klb})$
  - **Two restrictions:** a single value attribute per relation, and unique composite indexes per relation ➔ **single value per tensor cell**

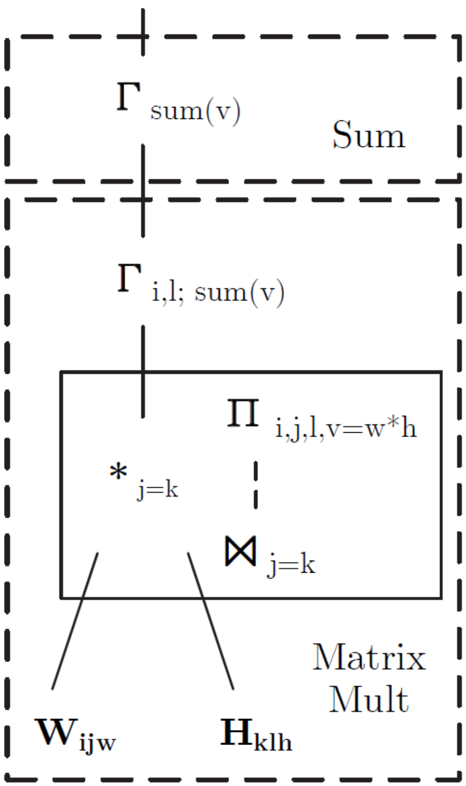
- **Example SP Plan**
  - **sum(W H)**



Sum-Product Optimization, cont.

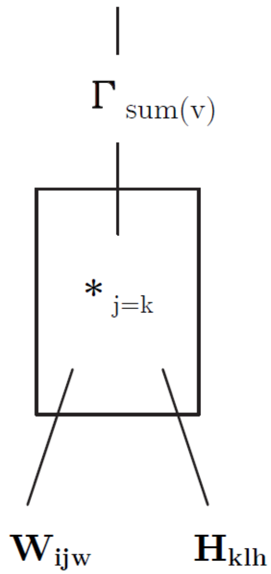
- **Example SP Plan Rewrites**
  – W:= 200K x 100, H:= 100 x 200K

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



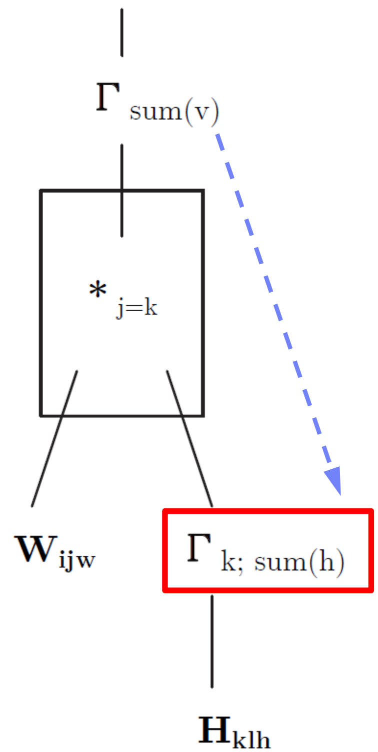
8.04 TFLOPs

Aggregation Elimination



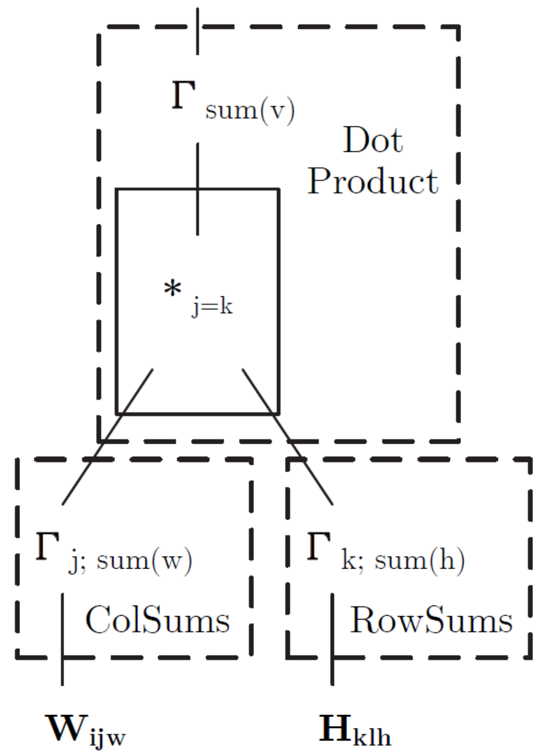
8 TFLOPs

Sum-Product (distributive law)



60 MFLOPs

Sum-Product (distributive law)

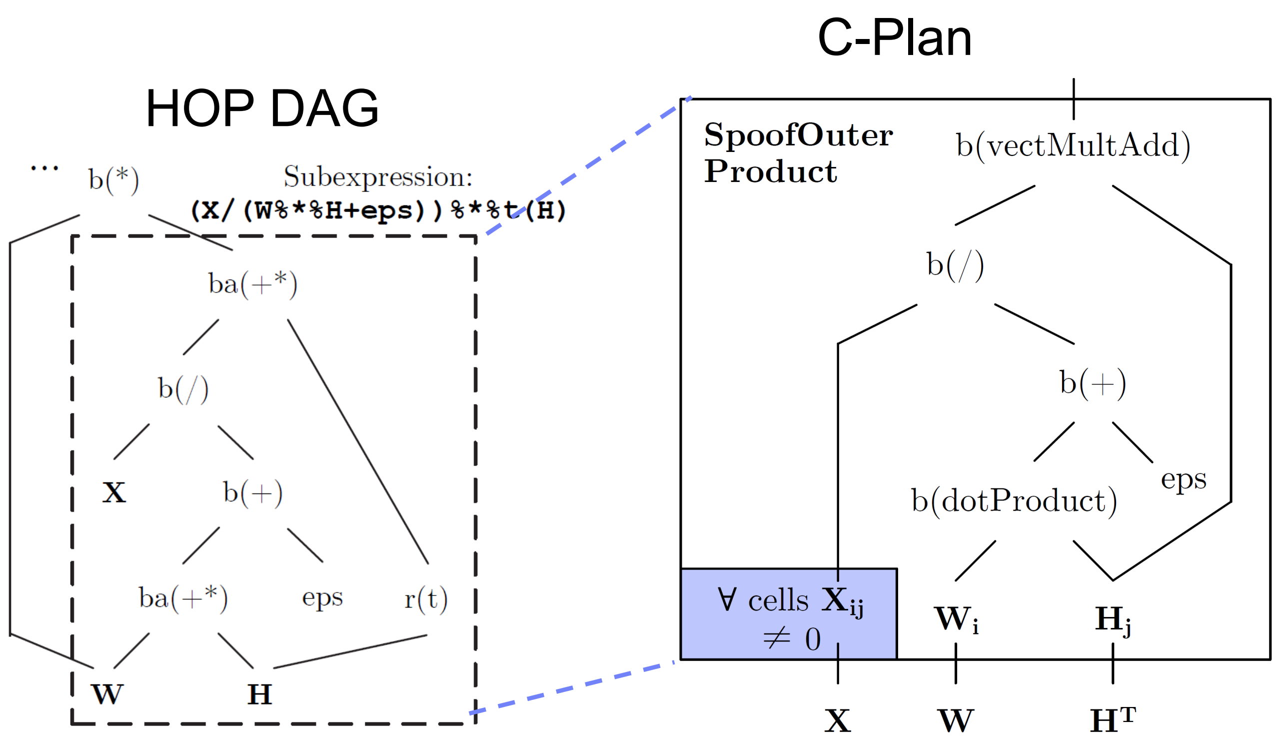


40 MFLOPs

Operator Fusion

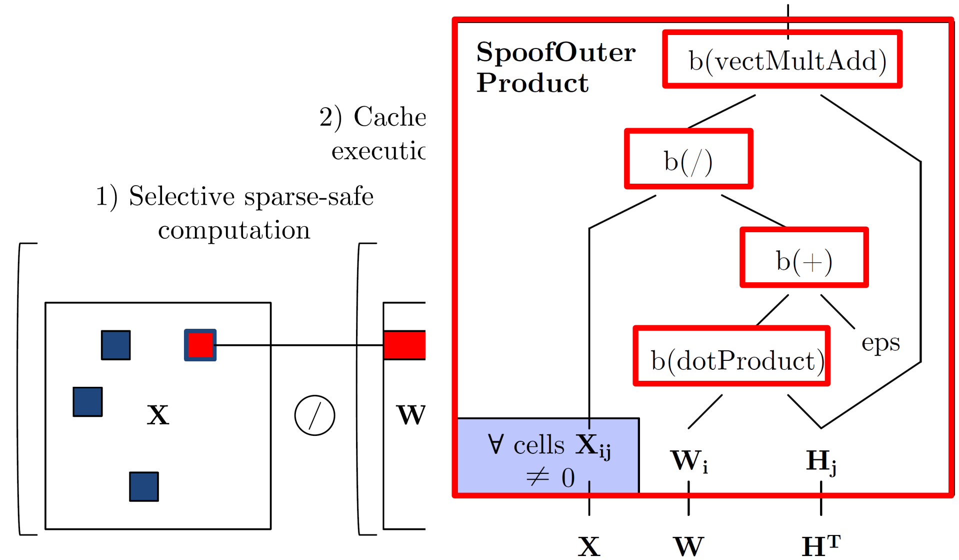
- **C-Plan Representation**
  - **Hybrid approach:** hand-coded operator skeletons with custom body code
    ➔ **Efficiency (data access, multi-threading) and flexibility**
  - **Template C-Nodes:** generic fused operator skeletons (w/ data binding)
    e.g., `SpoofOuterProduct`, `SpoofCellwise`, `SpoofRowAggregate`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

- **Example C-Plan**
  - **$(X / (W H + \varepsilon))\, H^T$**
    (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**




```
public final class TMP5 extends SpoofOuterProduct {
    public TMP5() {
        _type = OuterProductType.RIGHT;
    }
    protected void exec(double a, double[] b, int bi,
        double[] c, int ci,..., double[] d, int di, int k)
    {
        double TMP1 = dotProduct(b, c, bi, ci, k);      // WH
        double TMP2 = TMP1 + 1.0E-15;                   // +eps
        double TMP3 = a / TMP2;                         // X/
        vectMultiplyAdd(TMP3, c, d, ci, di, k);         // t(H)
    }
}
```

Custom body code

Experimental Setting

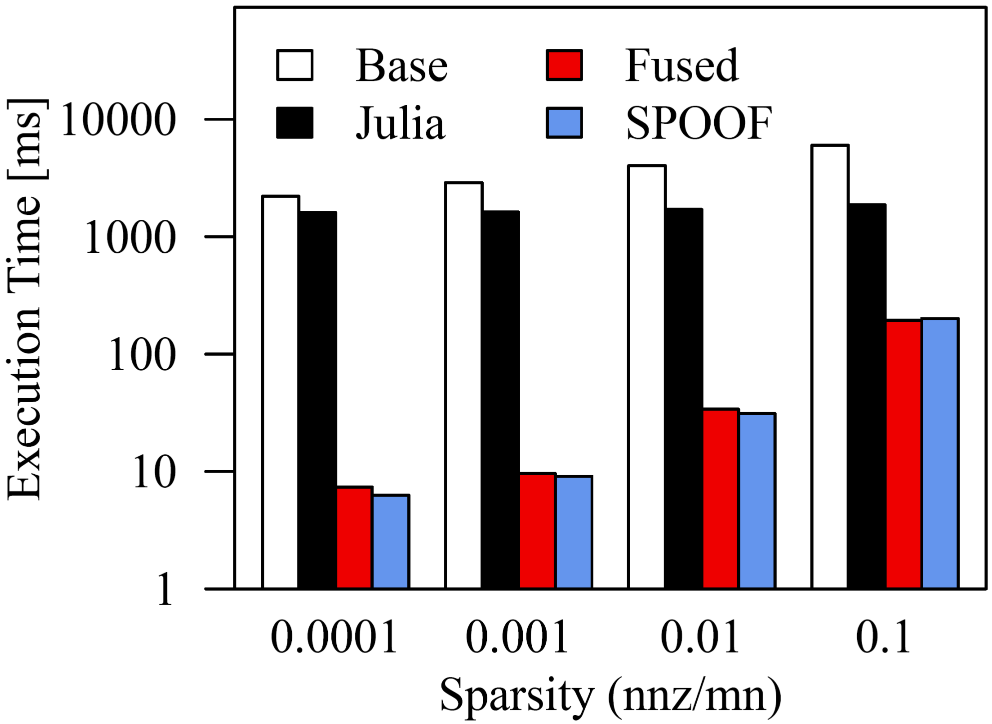
- **Cluster Setup**
  - 1 head node (2x4 Intel E5530, 64GB RAM), and
    6 worker nodes (2x6 Intel E5-2440, 96GB RAM, 12x2TB disks)
  - Spark 1.5.2 with 6 executors (24 cores, 60GB), 30GB driver memory

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

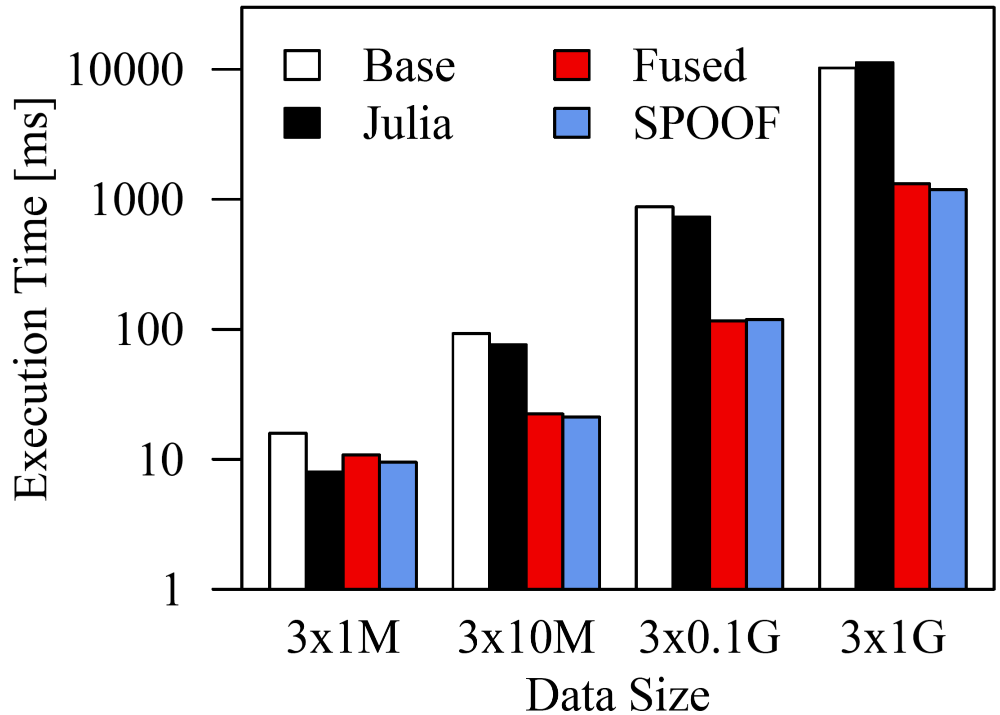
- **Selected Baselines**
  - Apache SystemML 0.10 (May 2016): **Base, Fused, SPOOF**
  - **Julia** 0.5 (Sep 2016) w/ LLVM-based just-in-time compiler

Micro Benchmarks: Operations Performance
(@ single worker node)

$(\mathbf{X} \,/\, (\mathbf{W}\,\mathbf{H} + \varepsilon))\,\mathbf{H}^{T}$, (PNMF)
10K x 10K, k=100, Multi-threaded



sum($\mathbf{X} \odot \mathbf{Y} \odot \mathbf{Z}$), (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

Dataset	Base	Fused	SPOOF
10K x 10K, 0.001	**251 s**	**6 s**	9 s
25K x 25 K, 0.001	**4,748 s**	**9 s**	11 s
200K x 200K, 0.001	**>24h**	**121 s**	125 s

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

Dataset	Base	Fused	SPOOF
100K x 10, 1.0 (8MB)	3 s	**3 s**	5 s
1M x 10, 1.0 (80MB)	9 s	**7 s**	8 s
10M x 10, 1.0 (800MB)	50 s	34 s	**17 s**
100M x 10, 1.0 (8GB)	525 s	320 s	**114 s**

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

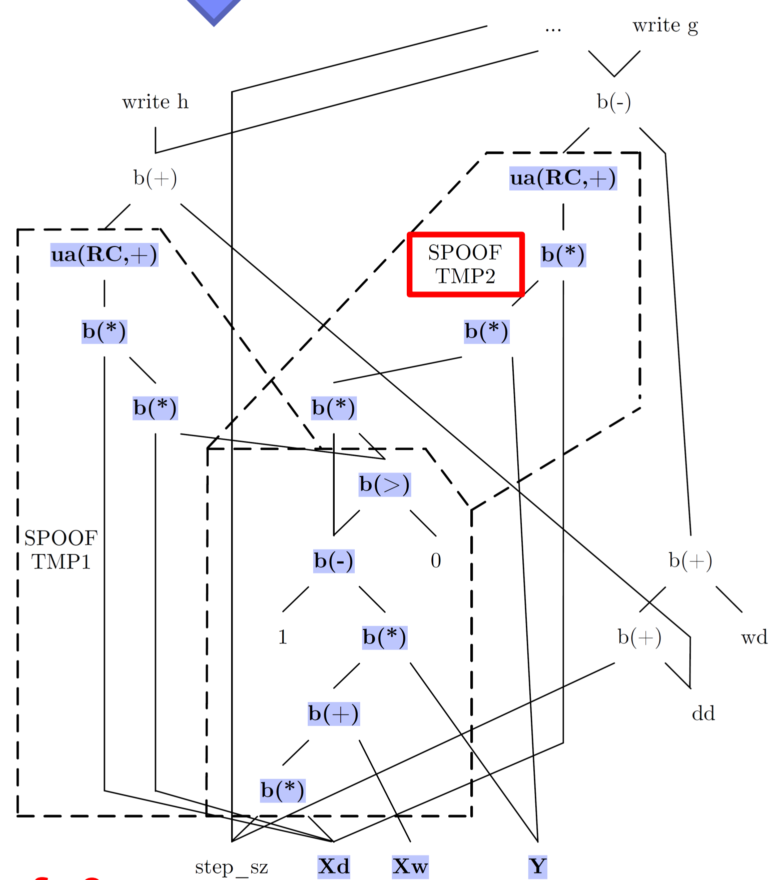
Website: http://systemml.apache.org/

Sources: https://github.com/apache/incubator-systemml

Backup: Operator Fusion, cont.

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

```
out = 1 - Y * (Xw + step_sz*Xd);
sv = (out > 0);
out = out * sv;
g = wd+step_sz*dd - sum(out*Y*Xd);
h = dd + sum(Xd*sv*Xd);
step_sz = step_sz - g/h;
```




```
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

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 / recompilations

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

Statistic	Spoof no PC	Spoof constant PC	Spoof CSLH
Execution time	**49.29 s**	19.87 s	**14.48 s**
PC hit rates	0 / 462	388 / 462	**449 / 462**
Javac compile time (sync)	**34.45 s**	6.88 s	**1.97 s**
JIT compile time (async)	25.36 s	18.84 s	**10.50 s**