AWARE: Workload-aware, Redundancy-exploiting Linear Algebra

Sebastian Baunsgaard & Matthias Boehm

Last updated — June 18, 2023

Matrix Compression

Lossy

- Main stream ML Systems
- Specialized data types
- Associated with trial and error

Matrix Compression

Lossy

- Main stream ML Systems
- Specialized data types
- Associated with trial and error

Lossless CLA

Matrix Compression

Lossy

- Main stream ML Systems
- Specialized data types
- Associated with trial and error

Lossless CLA

- Lightweight database compression
- Preaggregation Cocoding
- Correct results

Matrix Compression

Lossy

- Main stream ML Systems
- Specialized data types
- Associated with trial and error

Lossless CLA

- Lightweight database compression
- Preaggregation Cocoding
- Correct results

Optimization Goal

- Normal: **Compression Ratio**
- We: **Workload Time**

Available

SystemDS

https://github.com/apache/systemds

Reproducibility

https://github.com/damslab/reproducibility

Workload Awareness

User Script:

```
X = read("data/X")
y = read("data/y")

X = scale(X,TRUE,TRUE)
w = l2svm(X,y,TRUE,
      1e-9,1e-3,100)

write(w,"data/wXy")
```

Workload Awareness

User Script:

```
X = read("data/X")
y = read("data/y")

X = scale(X,TRUE,TRUE)
w = l2svm(X,y,TRUE,
        1e-9,1e-3,100)

write(w,"data/wXy")
```

```
if(shift)
 X = X - colMeans(X)
if(scale)
 X = X / colSds(X)
```

Built-in Functions:

```
if(intercept)
 X = cbind(X,ones)
while(conto & i<maxi) {
 Xd = X %*% s
 while(conti) {
  out = 1-y*(Xw+sz*Xd)
  sz = sz - g/h; # ...
 }
 g_new = t(X) %*% (out*y)
}
```

Workload Awareness

User Script:

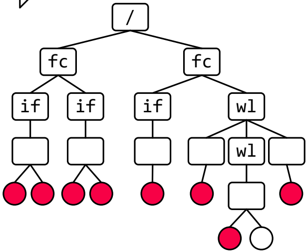
```
X = read("data/X")
y = read("data/y")

X = scale(X,TRUE,TRUE)
w = l2svm(X,y,TRUE,
      1e-9,1e-3,100)

write(w,"data/wXy")
```

```
if(shift)
 X = X - colMeans(X)
if(scale)
 X = X / colSds(X)
```

Built-in Functions:

```
if(intercept)
 X = cbind(X,ones)
while(conto & i<maxi) {
 Xd = X %*% s
 while(conti) {
  out = 1-y*(Xw+sz*Xd)
  sz = sz - g/h; # ...
 }
 g_new = t(X) %*% (out*y)
}
```

Workload Tree



Workload Awareness

User Script:

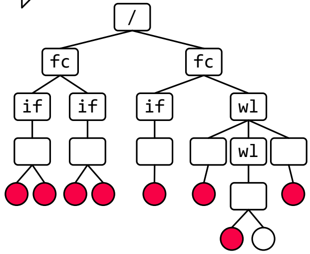
```
X = read("data/X")
y = read("data/y")

X = scale(X,TRUE,TRUE)
w = l2svm(X,y,TRUE,
      1e-9,1e-3,100)

write(w,"data/wXy")
```

```
if(shift)
 X = X - colMeans(X)
if(scale)
 X = X / colSds(X)
```

Built-in Functions:

```
if(intercept)
 X = cbind(X,ones)
while(conto & i<maxi) {
 Xd = X %*% s
 while(conti) {
  out = 1-y*(Xw+sz*Xd)
  sz = sz - g/h; # ...
 }
 g_new = t(X) %*% (out*y)
}
```

Workload Tree

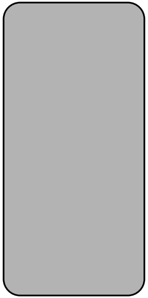


Cost Summary

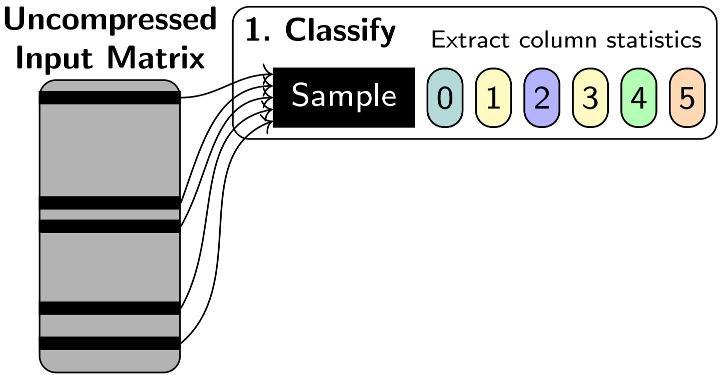
0	100	10	10	105	0

Compression Execution

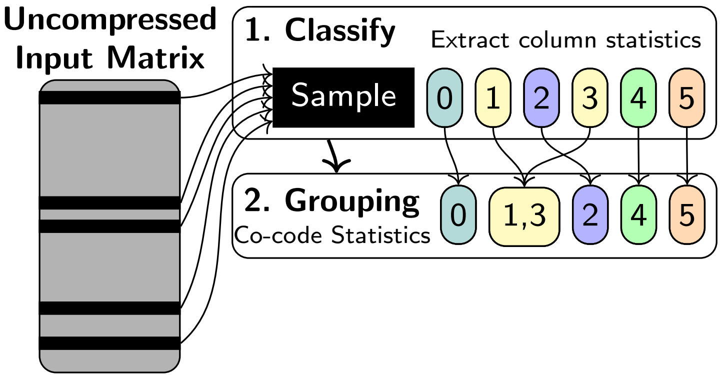
Uncompressed
Input Matrix



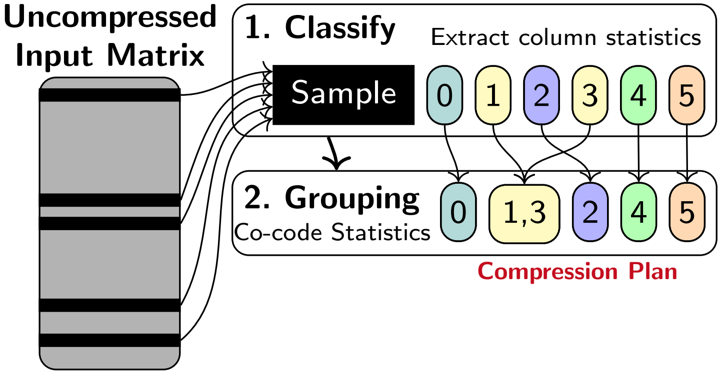
Compression Execution



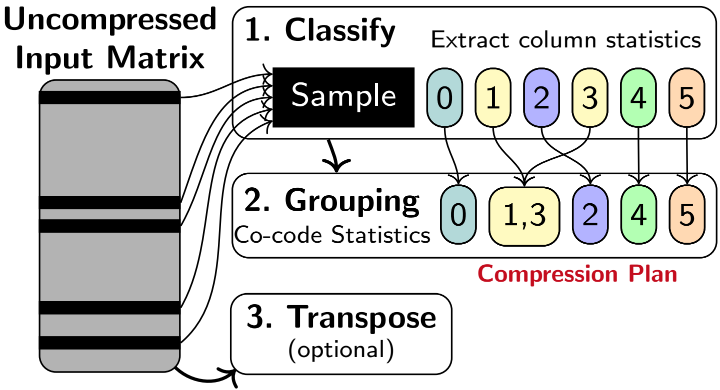
Compression Execution



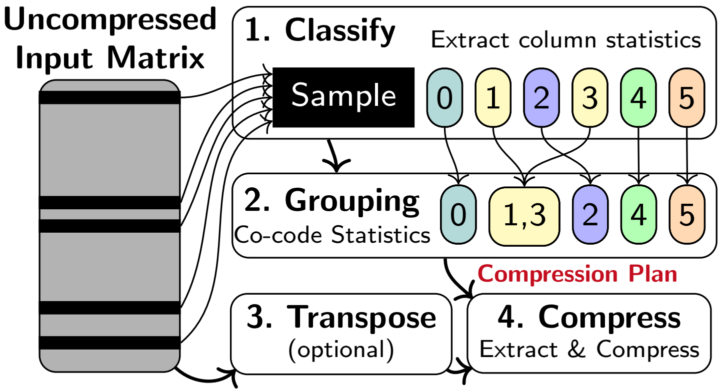
Compression Execution



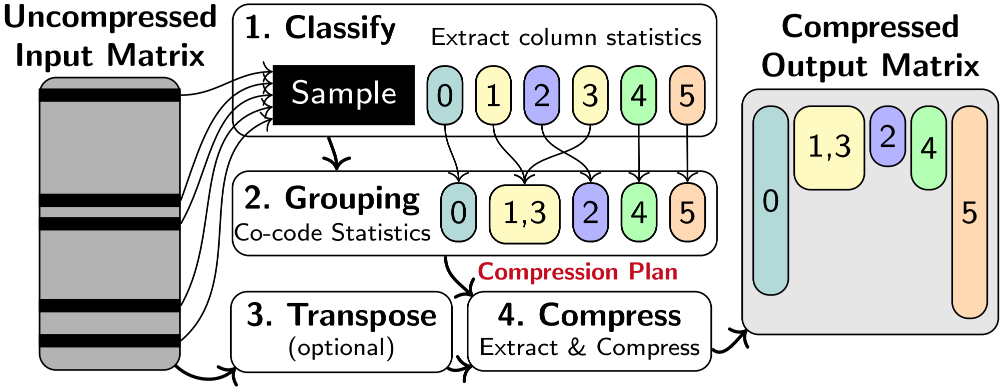
Compression Execution



Compression Execution



Compression Execution



Compression Example

Uncompressed

1	2	0	6	0	4
7	3	0	4	0	7
1	2	0	6	0	4
7	2	0	5	0	7
1	3	9	4	0	4
7	0	8.5	0	0	7
1	3	8.5	4	0	7
7	3	0	4	0	7
1	2	0	6	0	4
7	3	9	4	0	7

Compression Example

Uncompressed

1	2	0	6	0	4
7	3	0	4	0	7
1	2	0	6	0	4
7	2	0	5	0	7
1	3	9	4	0	4
7	0	8.5	0	0	7
1	3	8.5	4	0	7
7	3	0	4	0	7
1	2	0	6	0	4
7	3	9	4	0	7

Compressed

Compression Example

Uncompressed

1	2	0	6	0	4
7	3	0	4	0	7
1	2	0	6	0	4
7	2	0	5	0	7
1	3	9	4	0	4
7	0	8.5	0	0	7
1	3	8.5	4	0	7
7	3	0	4	0	7
1	2	0	6	0	4
7	3	9	4	0	7

Compressed

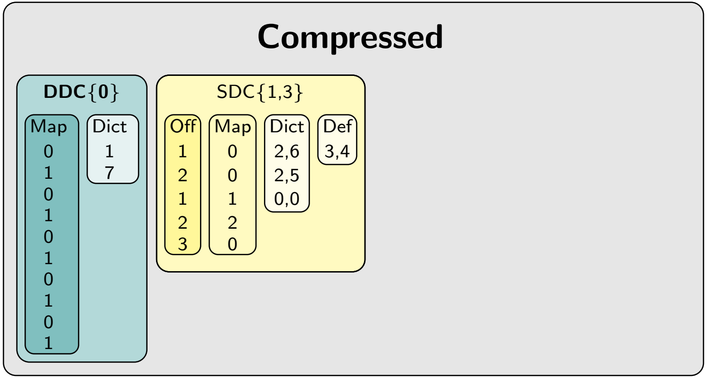
DDC{0}

Map	Dict
0	1
1	7
0	
1	
0	
1	
0	
1	
0	
1	

Compression Example

Uncompressed

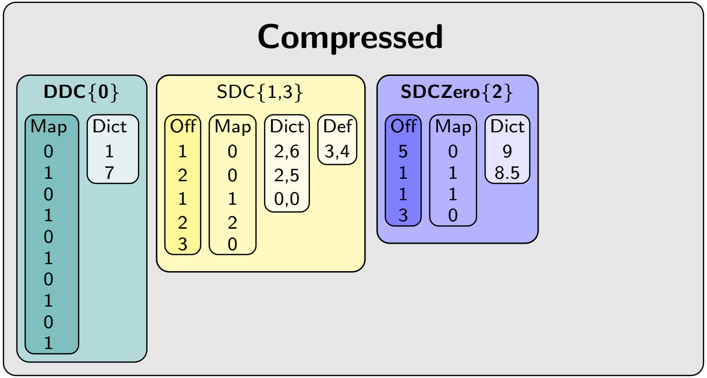
1	2	0	6	0	4
7	3	0	4	0	7
1	2	0	6	0	4
7	2	0	5	0	7
1	3	9	4	0	4
7	0	8.5	0	0	7
1	3	8.5	4	0	7
7	3	0	4	0	7
1	2	0	6	0	4
7	3	9	4	0	7



Compression Example

Uncompressed

1	2	0	6	0	4
7	3	0	4	0	7
1	2	0	6	0	4
7	2	0	5	0	7
1	3	9	4	0	4
7	0	8.5	0	0	7
1	3	8.5	4	0	7
7	3	0	4	0	7
1	2	0	6	0	4
7	3	9	4	0	7



Compression Example

Uncompressed

1	2	0	6	0	4
7	3	0	4	0	7
1	2	0	6	0	4
7	2	0	5	0	7
1	3	9	4	0	4
7	0	8.5	0	0	7
1	3	8.5	4	0	7
7	3	0	4	0	7
1	2	0	6	0	4
7	3	9	4	0	7

Compressed

DDC{0}

Map: 0, 1, 0, 1, 0, 1, 0, 1, 0, 1

Dict: 1, 7

SDC{1,3}

Off: 1, 2, 1, 2, 3

Map: 0, 0, 1, 2, 0

Dict: 2,6; 2,5; 0,0

Def: 3,4

SDCZero{2}

Off: 5, 1, 1, 3

Map: 0, 1, 1, 0

Dict: 9, 8.5

SDCSingle{5}

Off: 1, 2, 2, 4

Dict: 4

Def: 7

Difference CLA

	CLA	Aware
Co-Coding	$\mathcal{O}(m^2)$	
Column Group Encodings	4 (5)	
Materialization	Eager	
Optimization Objective	Data	
Matrix Multiplication	MV, VM	

Difference CLA

	CLA	Aware
Co-Coding	$\mathcal{O}(m^2)$	$\mathcal{O}(m)$
Column Group Encodings	4 (5)	
Materialization	Eager	
Optimization Objective	Data	
Matrix Multiplication	MV, VM	

Difference CLA

	CLA	Aware
Co-Coding	$\mathcal{O}(m^2)$	$\mathcal{O}(m)$
Column Group Encodings	4 (5)	7 (327)
Materialization	Eager	
Optimization Objective	Data	
Matrix Multiplication	MV, VM	

Difference CLA

	CLA	Aware
Co-Coding	$\mathcal{O}(m^2)$	$\mathcal{O}(m)$
Column Group Encodings	4 (5)	7 (327)
Materialization	Eager	Deferred
Optimization Objective	Data	
Matrix Multiplication	MV, VM	

Difference CLA

	CLA	Aware
Co-Coding	$\mathcal{O}(m^2)$	$\mathcal{O}(m)$
Column Group Encodings	4 (5)	7 (327)
Materialization	Eager	Deferred
Optimization Objective	Data	Data & Ops
Matrix Multiplication	MV, VM	

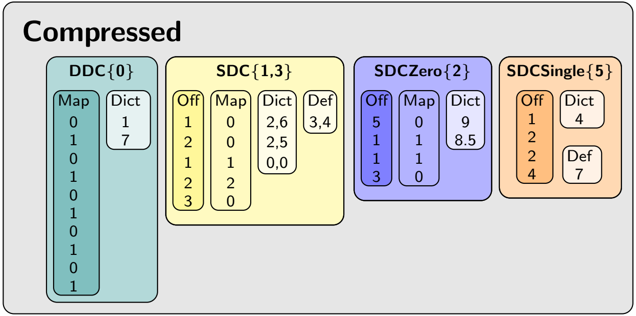
Difference CLA

	CLA	Aware
Co-Coding	$\mathcal{O}(m^2)$	$\mathcal{O}(m)$
Column Group Encodings	4 (5)	7 (327)
Materialization	Eager	Deferred
Optimization Objective	Data	Data & Ops
Matrix Multiplication	MV, VM	MV, VM, MM

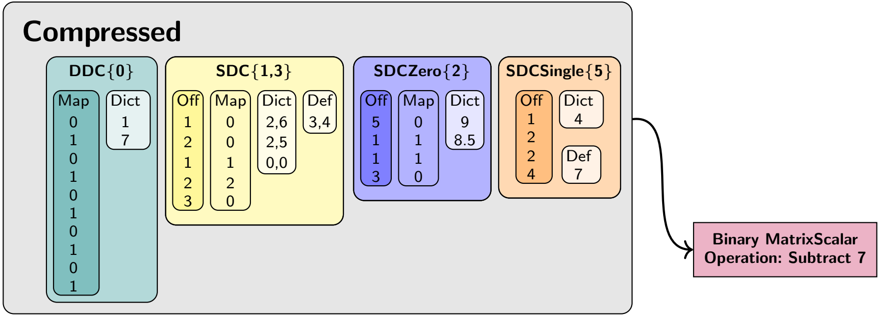
Difference CLA

Type	Description	CLA	Aware
CON	Constant or Empty Columns		✓
DDC	Dense Dictionary Coding	✓	✓
OLE	Offset-list Encoding	✓	(✓)
FOR	Frame of Reference		✓
RLE	Run-length Encoding	✓	(✓)
SDC	Sparse Dictionary Coding		✓
UC	Uncompressed (dense/sparse)	✓	✓

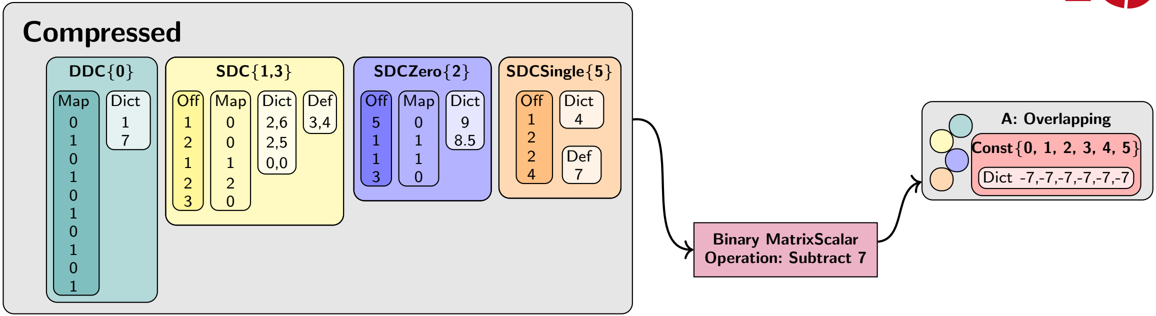
Operation Example



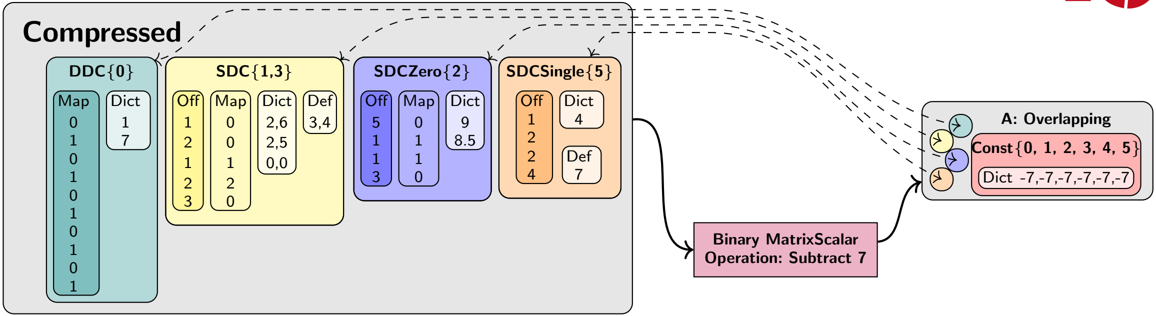
Operation Example



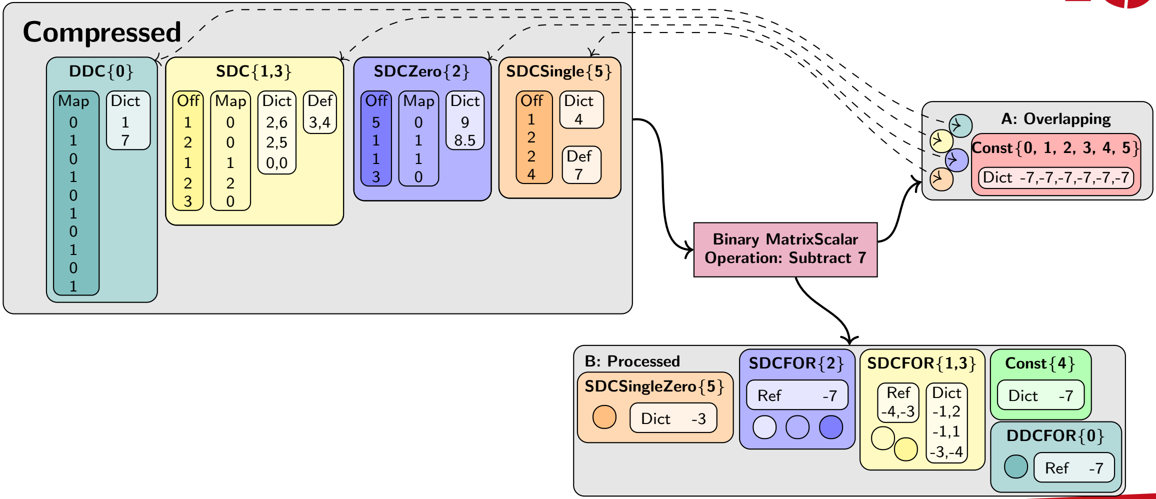
Operation Example



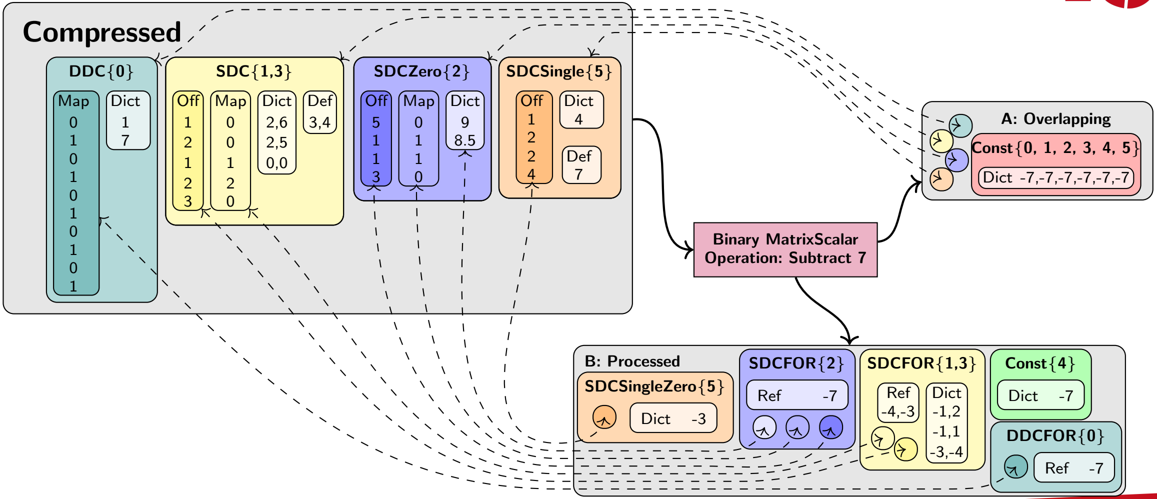
Operation Example



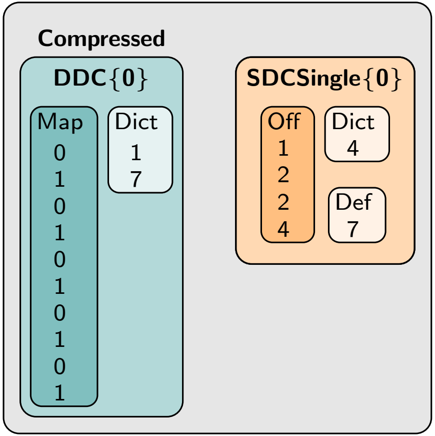
Operation Example



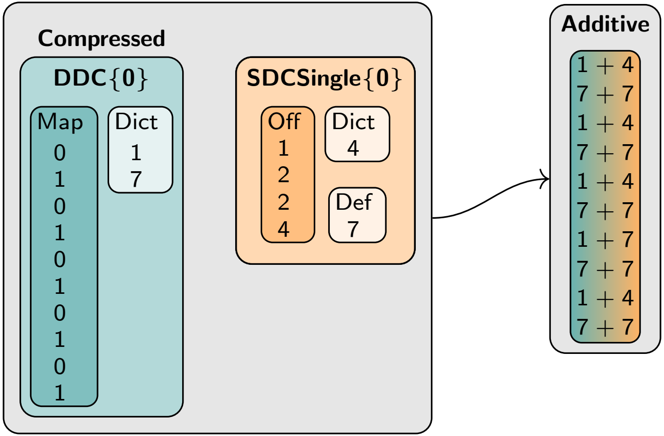
Operation Example



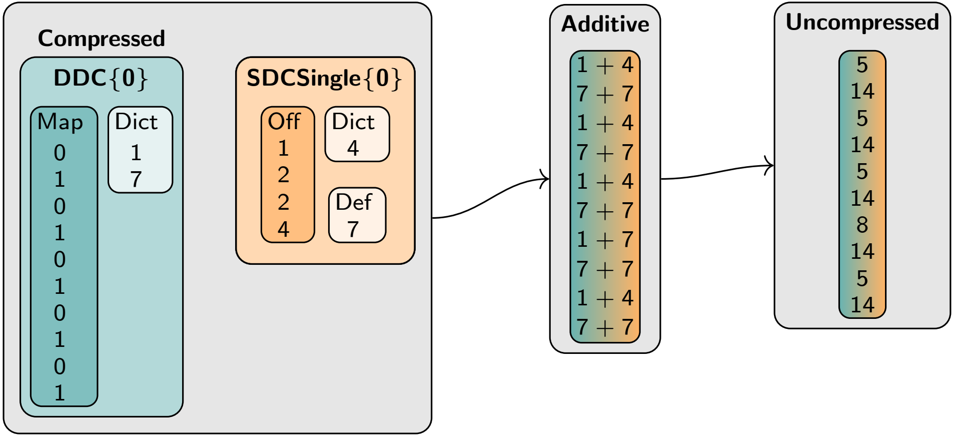
Overlapping Example



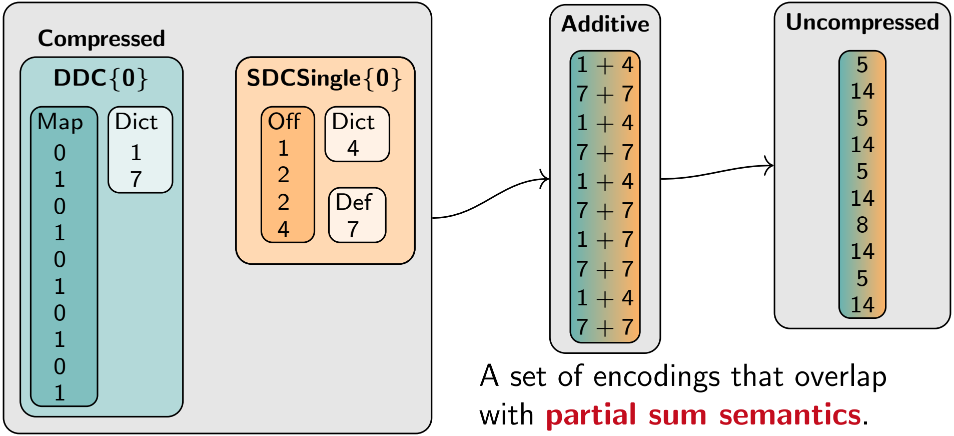
Overlapping Example



Overlapping Example

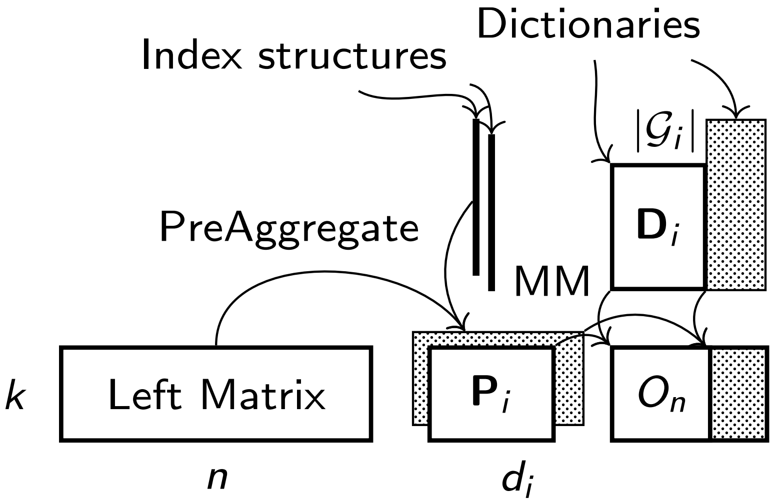


Overlapping Example

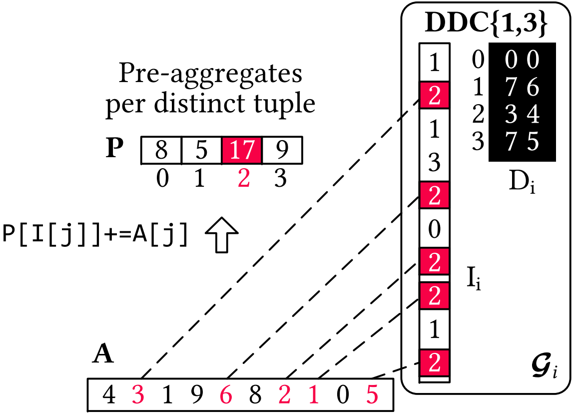


A set of encodings that overlap with **partial sum semantics**.

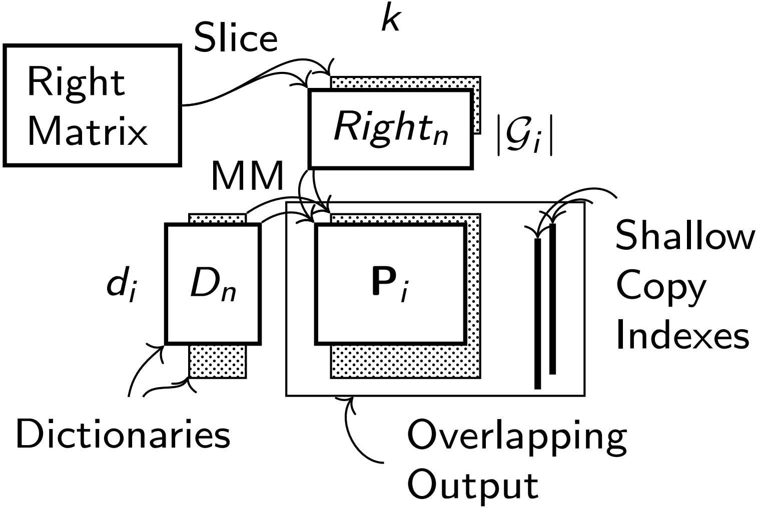
Left Matrix Multiplication



Preaggregate



Right Matrix Multiplication



Datasets

Datasets (*n* Rows, *m* Columns, *sp* Sparsity).

Dataset	*n* (nrow(**X**))	*m* (ncol(**X**))	*sp*	Size
Airline78	14,462,943	29	0.54	3.4 GB
Amazon	8,026,324	**2,330,066**	1.2e-6	1.22 GB
Covtype	581,012	54	0.22	127 MB
Mnist1m	1,000,000	784	0.25	2.46 GB
US Census	2,458,285	68 **(378)**	0.43 (0.18)	1.34 GB
US Census 128x	314,660,480	68 (378)	0.43 (0.18)	289.5 GB

Million column dataset and **high cocoding potential** scale up dataset

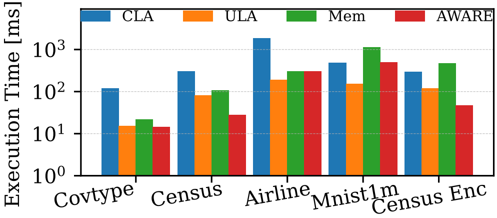
Compression Time

Local Compression Times [Seconds] and Ratios.

Dataset	CLA		Aware-Mem		Aware	
	time	ratio	time	ratio	time	ratio
Airline78	9.34 sec	10.22	1.74 sec	8.61	2.08 sec	7.94
Amazon	37.6 *hours*	*Crash*	8.54 sec	1.73	3.77 sec	*Abort*
Covtype	1.10 sec	13.79	0.84 sec	14.24	1.23 sec	13.99
Mnist1m	7.25 sec	7.14	4.57 sec	6.09	17.50 sec	4.41
US Census	5.15 sec	35.38	1.16 sec	29.60	1.15 sec	27.35
US Census Enc	27.48 sec	41.03	5.78 sec	38.46	6.54 sec	29.46

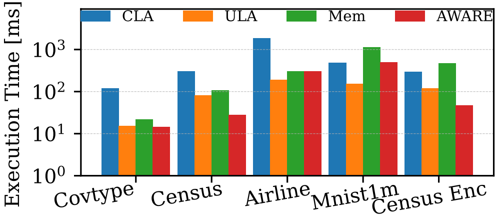
Intelligent compression abort based on workload and
Faster compression

Matrix Multiplication

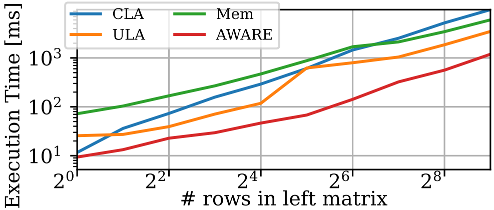


(a) 16 row LMM

Matrix Multiplication

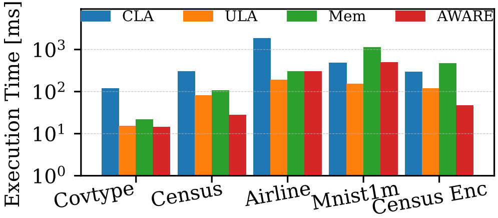


(e) 16 row LMM

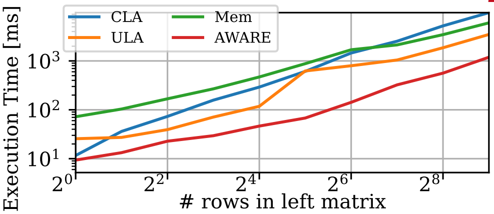


(f) LMM Census Enc Scaling

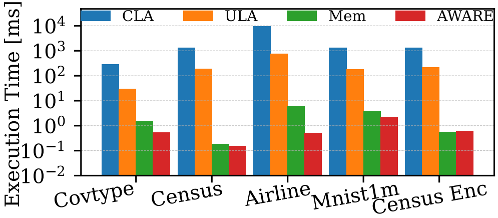
Matrix Multiplication



(i) 16 row LMM

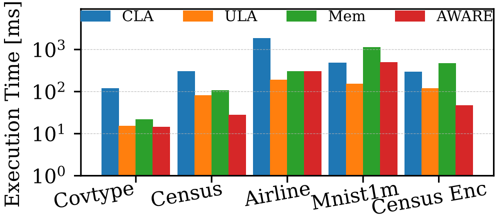


(j) LMM Census Enc Scaling

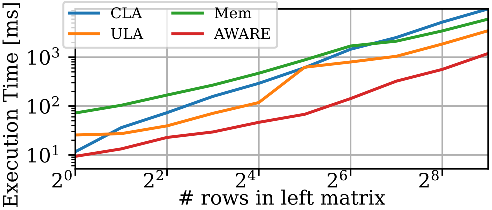


(k) 16 col RMM

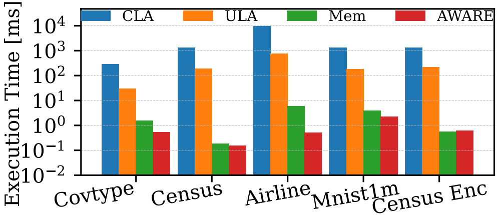
Matrix Multiplication



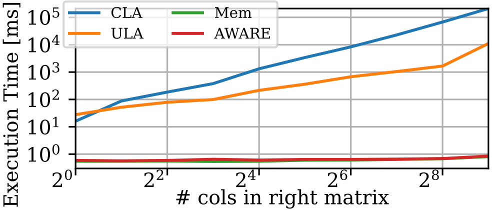
(m) 16 row LMM



(n) LMM Census Enc Scaling

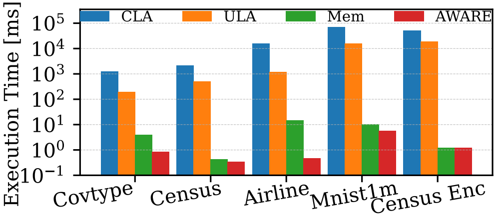


(o) 16 col RMM



(p) RMM Census Enc Scaling

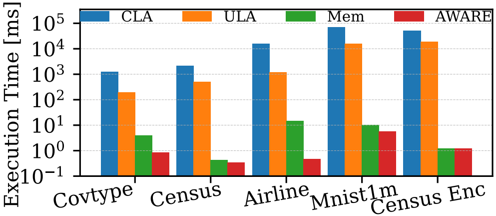
Operation Sequences



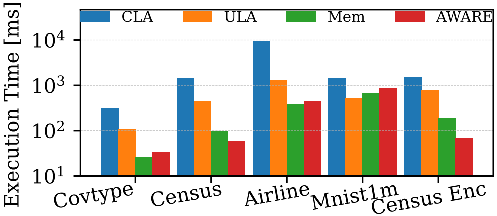
(q) Scale and Shift Sparse

Operation sequences maintain **compressed formats** and

Operation Sequences



(s) Scale and Shift Sparse



(t) Euclidean MinDist 16 Points Dense

Operation sequences maintain **compressed formats** and
Decompressing sequences are competitive or **faster** than uncompressed

Local End-to-End Algorithms

Workload-awareness on Local End-to-End Algorithms (Data: US Census Enc)

	ULA	Aware-Mem		Aware	
	Time	Comp	Time	Comp	Time
K-Means	51.6 sec	4.2 sec	46.2 sec	6.2 sec	27.1 sec
PCA	12.7 sec	4.0 sec	10.4 sec	6.0 sec	9.0 sec
MLogReg	32.0 sec	4.5 sec	32.5 sec	7.2 sec	26.0 sec
lmCG	19.8 sec	5.0 sec	20.7 sec	6.4 sec	18.6 sec
lmDS	15.6 sec	5.7 sec	15.5 sec	6.1 sec	14.3 sec
L2SVM	38.9 sec	6.5 sec	45.2 sec	6.2 sec	36.5 sec

AWARE is able to **amortize** the cost of **online compression** in short jobs.

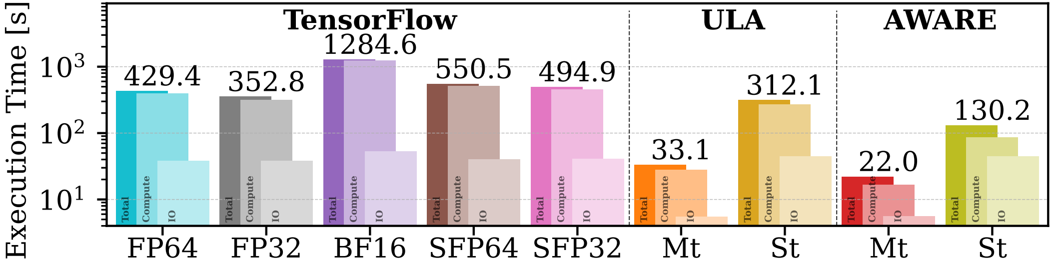
End-to-End Algorithms Hybrid Execution

[Seconds] (Data: US Census Enc, *D* .. Incl. Distributed Ops).

	K-Means		PCA		MLogReg	
	ULA	**Aware**	**ULA**	**Aware**	**ULA**	**Aware**
1x	51.6	(6) 27.1	12.7	(6) 9.4	32.0	(7) 26.0
8x	471.0	(26) 117.8	330.3	(26) 42.6	393.3	(29) 88.2
16x	^{D}484.3	(48) 183.9	^{D}76.3	(47) 67.5	^{D}570.3	(58) 144.2
32x	^{D}1,491.6	^{D}1,496.3	^{D}70.3	^{D}61.2	^{D}671.5	^{D}629.9
128x	^{D}17,819.0	^{D}6,298.0	^{D}137.0	^{D}140.3	^{D}3,502.9	^{D}1,710.6
*128x	^{D}33,039.0	^{D}11,616.0	^{D}269.0	^{D}259.0	^{D}50,998.0	^{D}8,599.6

AWARE keeps operations local, since they **fit in memory**.

TensorFlow Comparison



ImCG (Data: US Census Enc)

Our **baseline is competitive**, and **AWARE is the fastest**.

Grid Search

GridSearch MLogReg (Data: US Census Enc).

ULA	Aware-Mem	Aware
274.3 sec	238.1 sec	**92.6 sec**

Aware reduce **memory bandwidth** and **reduce compute** requirements.

Conclusions

- Optimize for **execution cost** not size of data
- Use Compressed Linear Algebra for **redundancy exploitation**,
  it is the natural next step from **sparsity exploitation**
- Workload awareness exploit **workload and data** characteristics together

Conclusions

- Optimize for **execution cost** not size of data
- Use Compressed Linear Algebra for **redundancy exploitation**,
  it is the natural next step from **sparsity exploitation**
- Workload awareness exploit **workload and data** characteristics together

Future Work

- Compressed **feature transformation** of heterogeneous data
- Orthogonal **lossy compression**