DIZK
A Distributed Zero Knowledge Proof System

Howard Wu, Wenting Zheng, Alessandro Chiesa, Raluca Ada Popa, Ion Stoica
University of California Berkeley

27th USENIX Security Symposium
August 16, 2018
Zero Knowledge Proof

[GMR 85]
Zero Knowledge Proof

[GMR 85]

Prover

Verifier
Zero Knowledge Proof

[GMR 85]

Prover

\[ F \] function

\[ y \] claimed output

Verifier

\[ F \] function

\[ y \] claimed output
Zero Knowledge Proof

I know $x$ s.t. $F(x) = y$

<table>
<thead>
<tr>
<th>Prover</th>
<th>Verifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F$</td>
<td>function</td>
</tr>
<tr>
<td>$y$</td>
<td>claimed output</td>
</tr>
<tr>
<td>$x$</td>
<td>private input</td>
</tr>
</tbody>
</table>
Zero Knowledge Proof

[GMR 85]

I know $x$ s.t. $F(x) = y$

Prover

$F$ | function
---|---
$y$ | claimed output
$x$ | private input

Verifier

$F$ | function
---|---
$y$ | claimed output
zkSNARK

Prover

F | function
---|---
y | claimed output
x | private input

Verifier

F | function
---|---
y | claimed output

I know x s.t. y = F(x)
zkSNARK

Prover

<table>
<thead>
<tr>
<th>F</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>claimed output</td>
</tr>
<tr>
<td>x</td>
<td>private input</td>
</tr>
</tbody>
</table>

Verifier

<table>
<thead>
<tr>
<th>F</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>claimed output</td>
</tr>
</tbody>
</table>

I know $x$ s.t. $y = F(x)$

Non-interactive
zkSNARK

Prover

<table>
<thead>
<tr>
<th>F</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>claimed output</td>
</tr>
<tr>
<td>x</td>
<td>private input</td>
</tr>
</tbody>
</table>

Verifier

<table>
<thead>
<tr>
<th>F</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>claimed output</td>
</tr>
</tbody>
</table>

I know $x$ s.t. $y = F(x)$

Succinct

Non-interactive
preprocessing **zkSNARK**

I know $x$ s.t. $y = F(x)$

**Prover**

- $F$ function
- $y$ claimed output
- $x$ private input

**Verifier**

- $F$ function
- $y$ claimed output

**Setup**

- $F$ function

**Proving Key**

**Verification Key**

**Succinct**

**Non-interactive**
Application #1

[BCGGMTV 14]

Alice

Bob
Application #1

[BCGGMTV 14]
Application #1

[BCGGMTV 14]

Anonymous P2P Payments
Application #2
Application #2

Smart Contracts
Application #2

Smart Contracts

\[ F(x) = y \]
Application #2

Smart Contracts

\[ F(x) = y \]

\[ F(x) = y? \]

\[ F(x) = y? \]

\[ F(x) = y? \]
Application #2

Smart Contracts

$F(x) = y$?
Application #2

Smart Contracts

π

π

π

π

F(x) = y?
Good News & Bad News
Good News & Bad News

Circuit size (logarithmic)

$10^0 \quad 10^1 \quad 10^2 \quad 10^3 \quad 10^4 \quad 10^5 \quad 10^6 \quad 10^7 \quad 10^8 \quad 10^9$
Good News & Bad News

Application #1 (Private payments) - $10^6$ gates
Good News & Bad News

Application #1
(Private payments)

10^6 gates

Circuit size (logarithmic)
Good News & Bad News

Application #1
(Private payments)

Application #2
(Typical smart contract execution)

10^6 gates

10^8 gates

Circuit size (logarithmic)
Good News & Bad News

- Application #1 (Private payments): $10^6$ gates
- Application #2 (Typical smart contract execution): $10^8$ gates
- Application #2 (Large smart contract execution): $10^9$ gates

Circuit size (logarithmic)
Good News & Bad News

Application #1
(Private payments)

10^6 gates

Application #2
(Typical smart contract execution)

10^8 gates

Application #2
(Large smart contract execution)

10^9 gates

monolithic zkSNARK runs out of memory

Circuit size (logarithmic)
Good News & Bad News

- **Application #1** (Private payments)
  - $10^6$ gates

- **Application #2** (Typical smart contract execution)
  - $10^8$ gates

- **Application #2** (Large smart contract execution)
  - $10^9$ gates

A monolithic zkSNARK runs out of memory for circuits exceeding $10^7$ gates.
Good News & Bad News

Application #1 (Private payments)
- $10^6$ gates

Application #2 (Typical smart contract execution)
- $10^8$ gates

Application #2 (Large smart contract execution)
- $10^9$ gates

monolithic zkSNARK runs out of memory

Circuit size (logarithmic)

Feasible?
DIZK is a zero knowledge proof system that is:
DIZK is a zero knowledge proof system that is:

**DISTRIBUTED**

Enables the execution of a zkSNARK Setup and Prover across a **compute cluster**
DIZK is a zero knowledge proof system that is:

**DISTRIBUTED**
Enables the execution of a zkSNARK Setup and Prover across a compute cluster

**SCALABLE**
Reaches heretofore unreachable circuit sizes (up to billions of gates)
Double the number of machines → twice the circuit size
DIZK is a zero knowledge proof system that is:

**DISTRIBUTED**
Enables the execution of a zkSNARK Setup and Prover across a compute cluster

**SCALABLE**
Reaches heretofore unreachable circuit sizes (up to billions of gates)
Double the number of machines → twice the circuit size

**PARALLEL**
Speeds up the time it takes to generate a proof
Double the number of machines → twice as fast
Our Approach

MONOLITHIC zkSNARK
[Groth16]
Our Approach

**MONOLITHIC zkSNARK**

[Groth16]

**DISTRIBUTED zkSNARK**
Challenges

function F

Prover

input \( w \)

secret input \( w \)

Proving Key

Setup

Verification Key

Verifier

accept or reject

\( x \)

\( x \)

\( \pi \)
Challenges

function F

Prover

input x

secret input w

Proving Key

Setup

Verification Key

Verifier

accept or reject

π

x
Challenges

Prover

Verifier

Setup

Proving Key

Verification Key

\( w \)

secret input

input \( x \)

Prover

accept or reject

\( \pi \)
Challenges

1. Multiplying polynomials of degree that are in the billions

Setup

Prover

Verifier

input $x$

secret input $w$

Proving Key

Verification Key

accept or reject
Challenges

1. Multiplying polynomials of degree that are in the billions

2. Representing these polynomials as terabit-sized arrays

- Setup
- Prover
- Verifier

input $x$

secret input $w$

function $F$

Proving Key

Verification Key

accept or reject
Challenges

1. Multiplying polynomials of degree that are in the billions

2. Representing these polynomials as terabit-sized arrays

3. Accessing large pools of shared memory in complex patterns

Prover

Verifier

input $x$

secret input $w$

Proving Key

Verification Key

accept or reject

Setup

$F$
Challenges

1. Multiplying polynomials of degree that are in the billions

2. Representing these polynomials as terabit-sized arrays

3. Accessing large pools of shared memory in complex patterns

4. Synchronizing shared state that incurs significant network delays

Prover

Verifier

Setup

input $x$

secret input $w$

Proving Key

Verification Key

accept or reject
Challenges

1. Multiplying polynomials of degree that are in the billions

2. Representing these polynomials as terabit-sized arrays

3. Accessing large pools of shared memory in complex patterns

4. Synchronizing shared state that incurs significant network delays

Distributing a zkSNARK is challenging
DIZK Architecture
DIZK Architecture

Distributed Setup
DIZK Architecture

Distributed Setup

Distributed Instance Reduction

Distributed Lagrange

Distributed fixMSM
DIZK Architecture

- Distributed Setup
  - Distributed Instance Reduction
  - Distributed Lagrange
  - Distributed fixMSM

- Proving Key
- Verification Key
DIZK Architecture

Distributed Setup
- Distributed Instance Reduction
- Distributed Lagrange
- Distributed fixMSM

Distributed Prover
- Distributed Witness Reduction
- Distributed FFT
- Distributed varMSM

Verification Key

input

secret input
DIZK Architecture

Verifier

Verification Key

Accept or reject

Distributed Setup
- Distributed Instance Reduction
- Distributed Lagrange
- Distributed fixMSM

Proving Key

Distributed Prover
- Distributed Witness Reduction
- Distributed FFT
- Distributed varMSM

Distributed Instance Reduction

Distributed Lagrange

Distributed fixMSM

input

secret input

w

x

π
DIZK Architecture

Distributed Setup
- Distributed Instance Reduction
- Distributed Lagrange
- Distributed fixMSM

Distributed Prover
- Distributed Witness Reduction
- Distributed FFT
- Distributed varMSM

F

input

proving key

verification key

accept or reject

Verifier

secret input
DIZK Architecture

Distributed Setup

- Distributed Instance Reduction
- Distributed Lagrange
- Distributed fixMSM

Distributed Prover

- Distributed Witness Reduction
- Distributed FFT
- Distributed varMSM

Verifier

- input
- secret input
- Proving Key
- Verification Key
- accept or reject

\( F \)

\( x \)

\( w \)

\( \pi \)
DIZK Architecture

Verifier

Verification Key

input \( x \)

accept or reject

Distributed Setup

Distributed Instance Reduction

Distributed Lagrange

in the paper

Distributed FixMSM

in the paper

Distributed Prover

Distributed Witness Reduction

Distributed FFT

Distributed VarMSM

in the paper

in the paper

Proving Key

secret input \( w \)

Verification Key

in the paper

in the paper

in the paper
DIZK Architecture

- Distributed Setup
  - Distributed Instance Reduction
  - Distributed Lagrange
  - Distributed fixMSM

- Distributed Prover
  - Distributed Witness Reduction
  - Distributed FFT
  - Distributed varMSM

- input
  - secret input

- Proving Key
- Verification Key

Verifier
- accept or reject

in the paper
Witness Reduction
[GGPR 13]
Witness Reduction

[BGPR 13]

Billion gate circuit $\rightarrow$ Billion degree polynomial
Witness Reduction

[GGPR 13]

Billion gate circuit $\rightarrow$ Billion degree polynomial

\[
H(X) = \frac{\left( \sum_{i=0}^{N} A_i(X) z_i \right) \cdot \left( \sum_{i=0}^{N} B_i(X) z_i \right) - \left( \sum_{i=0}^{N} C_i(X) z_i \right)}{Z_D(X)}
\]
Witness Reduction

[Billion gate circuit $\rightarrow$ Billion degree polynomial]

$H(X) = \left( \sum_{i=0}^{N} A_i(X) z_i \right) \cdot \left( \sum_{i=0}^{N} B_i(X) z_i \right) - \left( \sum_{i=0}^{N} C_i(X) z_i \right)$

$Z_D(X)$
Witness Reduction

[GGPR 13]

Billion gate circuit $\rightarrow$ Billion degree polynomial

\[
H(X) = \left( \sum_{i=0}^{N} A_i(X) z_i \right) \cdot \left( \sum_{i=0}^{N} B_i(X) z_i \right) - \left( \sum_{i=0}^{N} C_i(X) z_i \right) = Z_D(X)
\]

$N = 10^9$
Witness Reduction

$[GGPR\ 13]$\n
Billion gate circuit $\rightarrow$ Billion degree polynomial

\[
H(X) = \frac{\left( \sum_{i=0}^{N} A_i(X) z_i \right) \cdot \left( \sum_{i=0}^{N} B_i(X) z_i \right) - \left( \sum_{i=0}^{N} C_i(X) z_i \right)}{Z_D(X)}
\]

$N = 10^9$
Witness Reduction

[BGPR 13]

Billion gate circuit $\rightarrow$ Billion degree polynomial

matrix $A = (A_0, \ldots, A_N)$

$z$ = vector of $N + 1$ field elements

$N = 10^9$
Strawman for $\sum_{i=0}^{N} A_i(X) z_i$
Strawman for \( \sum_{i=0}^{N} A_i(X) z_i \)

matrix \( \mathbf{a} = (a_0, \ldots, a_N) \)

\[
\begin{array}{c}
\mathbf{a}_0 \\
\mathbf{a}_1 \\
\vdots \\
\mathbf{a}_N \\
\end{array}
\]
Strawman for $\sum_{i=0}^{N} A_i(X) z_i$

matrix $a = (a_0, \ldots, a_N)$

vector $z$

\[
M
\begin{array}{c|c|c|c|c|c|c}
\hline
 & a_0 & \cdots & a_N \\
\hline
M & \hline
\end{array}
\]

\[
1
\begin{array}{c|c|c|c|c|c|c}
\hline
 & z_0 & \cdots & z_N \\
\hline
1 & \hline
\end{array}
\]
Strawman for $\sum_{i=0}^{N} A_i(X) z_i$
Strawman for \[ \sum_{i=0}^{N} A_i(X) z_i \]
Strawman for $\sum_{i=0}^{N} A_i(X) z_i$
Strawman for $\sum_{i=0}^{N} A_i(X) z_i$
Strawman for $\sum_{i=0}^{N} A_i(X) \ z_i$
Strawman for $\sum_{i=0}^{N} A_i(X) z_i$
Strawman for $\sum_{i=0}^{N} A_i(X) \; z_i$

(a$_i$, $z_i$) pairs

<table>
<thead>
<tr>
<th></th>
<th>(a$_{0,0}$, $z_0$)</th>
<th>(a$_{0,1}$, $z_0$)</th>
<th>(a$_{0,2}$, $z_0$)</th>
<th>\cdot \cdot \cdot</th>
<th>(a$_{0,M}$, $z_0$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a$_{1,0}$, $z_1$)</td>
<td>(a$_{1,1}$, $z_1$)</td>
<td>(a$_{1,2}$, $z_1$)</td>
<td>\cdot \cdot \cdot</td>
<td>(a$_{1,M}$, $z_1$)</td>
<td></td>
</tr>
<tr>
<td>\cdot</td>
<td>\cdot</td>
<td>\cdot</td>
<td>\cdot</td>
<td>\cdot</td>
<td>\cdot</td>
</tr>
<tr>
<td>\cdot</td>
<td>\cdot</td>
<td>\cdot</td>
<td>\cdot</td>
<td>\cdot</td>
<td>\cdot</td>
</tr>
<tr>
<td>(a$_{N,0}$, $z_N$)</td>
<td>(a$_{N,1}$, $z_N$)</td>
<td>(a$_{N,2}$, $z_N$)</td>
<td>\cdot \cdot \cdot</td>
<td>(a$_{N,M}$, $z_N$)</td>
<td></td>
</tr>
</tbody>
</table>
Strawman for \[ \sum_{i=0}^{N} A_i(X) z_i \]

<table>
<thead>
<tr>
<th></th>
<th>(991, ( z_0 ))</th>
<th>(681, ( z_0 ))</th>
<th>(1978, ( z_0 ))</th>
<th>( \cdots )</th>
<th>(517, ( z_0 ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, ( z_1 ))</td>
<td>(2476, ( z_1 ))</td>
<td>(0, ( z_1 ))</td>
<td>( \cdots )</td>
<td>( \cdots )</td>
<td>(0, ( z_1 ))</td>
</tr>
<tr>
<td>( \cdots )</td>
<td>( \cdots )</td>
<td>( \cdots )</td>
<td>( \cdots )</td>
<td>( \cdots )</td>
<td>( \cdots )</td>
</tr>
<tr>
<td>( \cdots )</td>
<td>( \cdots )</td>
<td>( \cdots )</td>
<td>( \cdots )</td>
<td>( \cdots )</td>
<td>( \cdots )</td>
</tr>
<tr>
<td>(0, ( z_N ))</td>
<td>(8629, ( z_N ))</td>
<td>(0, ( z_N ))</td>
<td>( \cdots )</td>
<td>( \cdots )</td>
<td>(0, ( z_N ))</td>
</tr>
</tbody>
</table>
Strawman for $\sum_{i=0}^{N} A_i(X) z_i$

<table>
<thead>
<tr>
<th>(991, $z_0$)</th>
<th>(681, $z_0$)</th>
<th>(1978, $z_0$)</th>
<th>$\cdot$</th>
<th>$\cdot$</th>
<th>(517, $z_0$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, $z_1$)</td>
<td>(2476, $z_1$)</td>
<td>(0, $z_1$)</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>(0, $z_1$)</td>
</tr>
<tr>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
</tr>
<tr>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
</tr>
<tr>
<td>(0, $z_N$)</td>
<td>(8629, $z_N$)</td>
<td>(0, $z_N$)</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>(0, $z_N$)</td>
</tr>
</tbody>
</table>

Almost sparse
Strawman for $\sum_{i=0}^{N} A_i(X) z_i$

Almost sparse

<table>
<thead>
<tr>
<th></th>
<th>(991, $z_0$)</th>
<th>(681, $z_0$)</th>
<th>(1978, $z_0$)</th>
<th>...</th>
<th>(517, $z_0$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, $z_1$)</td>
<td>(0, $z_1$)</td>
<td>(0, $z_1$)</td>
<td>...</td>
<td></td>
<td>(0, $z_1$)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>(0, $z_N$)</td>
<td>(0, $z_N$)</td>
<td>(0, $z_N$)</td>
<td>...</td>
<td></td>
<td>(0, $z_N$)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>(991, $z_0$)</th>
<th>(681, $z_0$)</th>
<th>(1978, $z_0$)</th>
<th>...</th>
<th>(517, $z_0$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, $z_1$)</td>
<td>(0, $z_1$)</td>
<td>(0, $z_1$)</td>
<td>...</td>
<td></td>
<td>(0, $z_1$)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>(0, $z_N$)</td>
<td>(0, $z_N$)</td>
<td>(0, $z_N$)</td>
<td>...</td>
<td></td>
<td>(0, $z_N$)</td>
</tr>
</tbody>
</table>
Strawman for $\sum_{i=0}^{N} A_i(X) z_i$

Almost sparse

<table>
<thead>
<tr>
<th>$A_i(X)$</th>
<th>$z_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(991, z_0)$</td>
<td>(0, $z_1$)</td>
</tr>
<tr>
<td>(681, $z_0$)</td>
<td>(2476, $z_1$)</td>
</tr>
<tr>
<td>(1978, $z_0$)</td>
<td>(0, $z_1$)</td>
</tr>
<tr>
<td>(517, $z_0$)</td>
<td>(0, $z_1$)</td>
</tr>
<tr>
<td>(8629, $z_N$)</td>
<td>(0, $z_N$)</td>
</tr>
<tr>
<td>(0, $z_N$)</td>
<td>(0, $z_N$)</td>
</tr>
</tbody>
</table>
Strawman for $\sum_{i=0}^{N} A_i(X) z_i$

Almost sparse

<table>
<thead>
<tr>
<th></th>
<th>(991, $z_0$)</th>
<th>(681, $z_0$)</th>
<th>(1978, $z_0$)</th>
<th>(517, $z_0$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, $z_1$)</td>
<td>(2476, $z_1$)</td>
<td>(0, $z_1$)</td>
<td>(0, $z_1$)</td>
<td>(0, $z_1$)</td>
</tr>
<tr>
<td>(0, $z_N$)</td>
<td>(8629, $z_N$)</td>
<td>(0, $z_N$)</td>
<td>(0, $z_N$)</td>
<td>(0, $z_N$)</td>
</tr>
</tbody>
</table>

fast  slow  fast  fast  fast  fast  fast  fast  fast

Almost sparse
Strawman for $\sum_{i=0}^{N} A_i(X) z_i$

Almost sparse

<table>
<thead>
<tr>
<th></th>
<th>(991, $z_0$)</th>
<th>(681, $z_0$)</th>
<th>(1978, $z_0$)</th>
<th>$\cdots$</th>
<th>(517, $z_0$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, $z_1$)</td>
<td>(2476, $z_1$)</td>
<td>(0, $z_1$)</td>
<td>$\cdots$</td>
<td>(0, $z_1$)</td>
<td></td>
</tr>
<tr>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td></td>
</tr>
<tr>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td></td>
</tr>
<tr>
<td>(0, $z_N$)</td>
<td>(8629, $z_N$)</td>
<td>(0, $z_N$)</td>
<td>$\cdots$</td>
<td>(0, $z_N$)</td>
<td></td>
</tr>
</tbody>
</table>
Strawman for $\sum_{i=0}^{N} A_i(X) z_i$

Almost sparse

<table>
<thead>
<tr>
<th></th>
<th>(991, $z_0$)</th>
<th>(681, $z_0$)</th>
<th>(1978, $z_0$)</th>
<th>· · ·</th>
<th>(517, $z_0$)</th>
</tr>
</thead>
</table>
| (0, $z_1$) | (2476, $z_1$) | (0, $z_1$) | · · · | (0, $z_1$) |}

Strawman for $\sum_{i=0}^{N} A_i(X) z_i$
Strawman for $\sum_{i=0}^{N} A_i(X) z_i$

| $(991, z_0)$ | $(681, z_0)$ | $(1978, z_0)$ | BAD | $\cdots$ | $\cdots$ | $(517, z_0)$ |
| $(0, z_1)$ | $(2476, z_1)$ | $(0, z_1)$ | OK | $\cdots$ | $\cdots$ | $(0, z_1)$ |
| $\cdots$ | $\cdots$ | $\cdots$ | OK | $\cdots$ | $\cdots$ | $\cdots$ |
| $\cdots$ | $\cdots$ | $\cdots$ | OK | $\cdots$ | $\cdots$ | $\cdots$ |
| $(0, z_N)$ | $(8629, z_N)$ | $(0, z_N)$ | OK | $\cdots$ | $\cdots$ | $(0, z_N)$ |
Off-the-shelf Approaches
Off-the-shelf Approaches

*Replicate* and *partition* the data so that the computation is *distributed evenly*. 
blockjoin

(Common technique to address data skew)
blockjoin

(Common technique to address data skew)

Replicated each entry for every machine
blockjoin

(Common technique to address data skew)

Replicated each entry for every machine
### blockjoin

(Common technique to address data skew)

<table>
<thead>
<tr>
<th>(a₀,0, z₀)</th>
<th>(a₀,1, z₀)</th>
<th>(a₀,2, z₀)</th>
<th>(a₀,3, z₀)</th>
<th>...</th>
<th>(a₀,M−1, z₀)</th>
<th>(a₀,M, z₀)</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>(a₀,0, N)</td>
<td>(a₀,1, N)</td>
<td>(a₀,2, N)</td>
<td>(a₀,3, N)</td>
<td></td>
<td>(a₀,M−1, N)</td>
<td>(a₀,M, N)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>(a,N,0, z₀)</td>
<td>(a,N,1, z₀)</td>
<td>(a,N,2, z₀)</td>
<td>(a,N,3, z₀)</td>
<td>...</td>
<td>(a,N,M−1, z₀)</td>
<td>(a,N,M, z₀)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>(a,N,0, N)</td>
<td>(a,N,1, N)</td>
<td>(a,N,2, N)</td>
<td>(a,N,3, N)</td>
<td></td>
<td>(a,N,M−1, N)</td>
<td>(a,N,M, N)</td>
</tr>
</tbody>
</table>

Common technique to address data skew
**blockjoin** *(N + 1) * (# partitions) replications*

(Common technique to address data skew)

<table>
<thead>
<tr>
<th>(a₀,₀, z₀)</th>
<th>(a₀,₁, z₀)</th>
<th>(a₀,₂, z₀)</th>
<th>(a₀,₃, z₀)</th>
<th>...</th>
<th>(a₀,M⁻¹, z₀)</th>
<th>(a₀,M, z₀)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(a₀,₀, zₙ)</td>
<td>(a₀,₁, zₙ)</td>
<td>(a₀,₂, zₙ)</td>
<td>(a₀,₃, zₙ)</td>
<td>...</td>
<td>(a₀,M⁻¹, zₙ)</td>
<td>(a₀,M, zₙ)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(aₙ,₀, z₀)</td>
<td>(aₙ,₁, z₀)</td>
<td>(aₙ,₂, z₀)</td>
<td>(aₙ,₃, z₀)</td>
<td>...</td>
<td>(aₙ,M⁻¹, z₀)</td>
<td>(aₙ,M, z₀)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Common technique to address data skew)
**blockjoin** *(N + 1) * (# partitions) replications*

(Common technique to address data skew)

<table>
<thead>
<tr>
<th>(a₀,₀, z₀₀)</th>
<th>(a₀,₁, z₀₀)</th>
<th>(a₀,₂, z₀₀)</th>
<th>(a₀,₃, z₀₀)</th>
<th>· · ·</th>
<th>(a₀,M⁻¹, z₀₀)</th>
<th>(a₀,M, z₀₀)</th>
</tr>
</thead>
<tbody>
<tr>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
</tr>
<tr>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
</tr>
<tr>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
</tr>
<tr>
<td>(a₀,₀, z₀ₙ)</td>
<td>(a₀,₁, z₀ₙ)</td>
<td>(a₀,₂, z₀ₙ)</td>
<td>(a₀,₃, z₀ₙ)</td>
<td>·</td>
<td>(a₀,M⁻¹, z₀ₙ)</td>
<td>(a₀,M, z₀ₙ)</td>
</tr>
<tr>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
</tr>
<tr>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
</tr>
<tr>
<td>(aₙ,₀, z₀₀)</td>
<td>(aₙ,₁, z₀₀)</td>
<td>(aₙ,₂, z₀₀)</td>
<td>(aₙ,₃, z₀₀)</td>
<td>·</td>
<td>(aₙ,M⁻¹, z₀₀)</td>
<td>(aₙ,M, z₀₀)</td>
</tr>
<tr>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
</tr>
<tr>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
</tr>
<tr>
<td>(aₙ,₀, z₀ₙ)</td>
<td>(aₙ,₁, z₀ₙ)</td>
<td>(aₙ,₂, z₀ₙ)</td>
<td>(aₙ,₃, z₀ₙ)</td>
<td>·</td>
<td>(aₙ,M⁻¹, z₀ₙ)</td>
<td>(aₙ,M, z₀ₙ)</td>
</tr>
</tbody>
</table>

Common technique to address data skew
Every partition is now dense, therefore the computation is uniform.

<table>
<thead>
<tr>
<th>(a_{0,0}, z_0)</th>
<th>(a_{0,1}, z_0)</th>
<th>(a_{0,2}, z_0)</th>
<th>(a_{0,3}, z_0)</th>
<th>...</th>
<th>(a_{0,M-1}, z_0)</th>
<th>(a_{0,M}, z_0)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(a_{0,0}, z_N)</td>
<td>(a_{0,1}, z_N)</td>
<td></td>
<td></td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(a_{N,0}, z_0)</td>
<td>(a_{N,1}, z_0)</td>
<td>(a_{N,2}, z_0)</td>
<td>(a_{N,3}, z_0)</td>
<td>...</td>
<td>(a_{N,M-1}, z_0)</td>
<td>(a_{N,M}, z_0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(a_{N,0}, z_N)</td>
<td>(a_{N,1}, z_N)</td>
<td>(a_{N,2}, z_N)</td>
<td>(a_{N,3}, z_N)</td>
<td>...</td>
<td>(a_{N,M-1}, z_0)</td>
<td>(a_{N,M}, z_N)</td>
</tr>
</tbody>
</table>
Every partition is now dense, therefore the computation is uniform.

(However, the table is huge and impractical to compute)
skewjoin

(Another common technique to address data skew. Source: Tresata)
skewjoin
(Another common technique to address data skew. Source: Tresata)

Compute usage statistics and replicate frequently-used entries for every machine.
skewjoin

(Another common technique to address data skew. Source: Tresata)

<table>
<thead>
<tr>
<th></th>
<th>(a₀,₀, z₀)</th>
<th>(a₀,₁, z₀)</th>
<th>(a₀,₂, z₀)</th>
<th>...</th>
<th>(a₀,M, z₀)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a₁,₀, z₁)</td>
<td>(a₁,₁, z₁)</td>
<td>(a₁,₂, z₁)</td>
<td>...</td>
<td></td>
<td>(a₁,M, z₁)</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>(aN,₀, zₙ)</td>
<td>(aN,₁, zₙ)</td>
<td>(aN,₂, zₙ)</td>
<td>...</td>
<td></td>
<td>(aN,M, zₙ)</td>
</tr>
</tbody>
</table>
Another common technique to address data skew. Source: Tresata

<table>
<thead>
<tr>
<th>$a_{0,0}$, $z_0$</th>
<th>$a_{0,1}$, $z_0$</th>
<th>$a_{0,2}$, $z_0$</th>
<th>...</th>
<th>$a_{0,M}$, $z_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_{1,0}$, $z_1$</td>
<td>$a_{1,1}$, $z_1$</td>
<td>$a_{1,2}$, $z_1$</td>
<td>...</td>
<td>$a_{1,M}$, $z_1$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$a_{N,0}$, $z_N$</td>
<td>$a_{N,1}$, $z_N$</td>
<td>$a_{N,2}$, $z_N$</td>
<td>...</td>
<td>$a_{N,M}$, $z_N$</td>
</tr>
</tbody>
</table>
skewjoin

(Another common technique to address data skew. Source: Tresata)
skewjoin == Strawman

(Another common technique to address data skew. Source: Tresata)
skewjoin

(Another common technique to address data skew. Source: Tresata)

<table>
<thead>
<tr>
<th>(a_{0,0}, z_0)</th>
<th>(a_{0,1}, z_0)</th>
<th>(a_{0,2}, z_0)</th>
<th>...</th>
<th>(a_{0,M}, z_0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a_{1,0}, z_1)</td>
<td>(a_{1,1}, z_1)</td>
<td>(a_{1,2}, z_1)</td>
<td>...</td>
<td>(a_{1,M}, z_1)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>(a_{N,0}, z_N)</td>
<td>(a_{N,1}, z_N)</td>
<td>(a_{N,2}, z_N)</td>
<td>...</td>
<td>(a_{N,M}, z_N)</td>
</tr>
</tbody>
</table>
**skewjoin**

(Another common technique to address data skew. Source: *Tresata*)

<table>
<thead>
<tr>
<th></th>
<th>(a₀,₀, z₀)</th>
<th>(a₀,₁, z₀)</th>
<th>(a₀,₂, z₀)</th>
<th>⋯</th>
<th>⋯</th>
<th>(a₀,M, z₀)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a₁,₀, z₁)</td>
<td>⋯</td>
<td>(a₁,₁, z₁)</td>
<td>(a₁,₂, z₁)</td>
<td>⋯</td>
<td>⋯</td>
<td>(a₁,M, z₁)</td>
</tr>
<tr>
<td>⋯</td>
<td>⋯</td>
<td>⋯</td>
<td>⋯</td>
<td></td>
<td></td>
<td>⋯</td>
</tr>
<tr>
<td>(aN,₀, z_N)</td>
<td>⋯</td>
<td>(aN,₁, z_N)</td>
<td>(aN,₂, z_N)</td>
<td>⋯</td>
<td>⋯</td>
<td>(aN,M, z_N)</td>
</tr>
</tbody>
</table>
skewjoin

(Another common technique to address data skew. Source: Tresata)

<table>
<thead>
<tr>
<th>(a₀,₀, z₀)</th>
<th>(a₀,₁, z₀)</th>
<th>(a₀,₂, z₀)</th>
<th>.</th>
<th>.</th>
<th>(a₀,M, z₀)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a₁,₀, z₁)</td>
<td>(a₁,₁, z₁)</td>
<td>(a₁,₂, z₁)</td>
<td>.</td>
<td>.</td>
<td>(a₁,M, z₁)</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>(a₉,₀, z₉)</td>
<td>(a₉,₁, z₉)</td>
<td>(a₉,₂, z₉)</td>
<td>.</td>
<td>.</td>
<td>(a₉,M, z₉)</td>
</tr>
</tbody>
</table>

z₀

z₁

z₉
skewJoin

(Another common technique to address data skew. Source: Tresata)

<table>
<thead>
<tr>
<th>(a_{0,0}, z_0)</th>
<th>(a_{0,1}, z_0)</th>
<th>(a_{0,2}, z_0)</th>
<th>\ldots</th>
<th>(a_{0,M}, z_0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a_{1,0}, z_1)</td>
<td>(a_{1,1}, z_1)</td>
<td>(a_{1,2}, z_1)</td>
<td>\ldots</td>
<td>(a_{1,M}, z_1)</td>
</tr>
<tr>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
</tr>
<tr>
<td>(a_{N,0}, z_N)</td>
<td>(a_{N,1}, z_N)</td>
<td>(a_{N,2}, z_N)</td>
<td>\ldots</td>
<td>(a_{N,M}, z_N)</td>
</tr>
</tbody>
</table>
skewjoin == blockjoin

(Another common technique to address data skew. Source: Tresata)
Tailored Approach
Tailored Approach

_Isolate_ and _transform_ the data so that the computation is _distributed evenly._
Tailored Approach — Part 1

Identify Dense Vectors

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>991</td>
<td>681</td>
<td>1978</td>
<td></td>
<td></td>
<td>517</td>
</tr>
<tr>
<td>0</td>
<td>2476</td>
<td>0</td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>8629</td>
<td>0</td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
## Tailored Approach — Part 1

### Identify Dense Vectors

<table>
<thead>
<tr>
<th>Density Count</th>
<th>991</th>
<th>681</th>
<th>1978</th>
<th>...</th>
<th>517</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>0</td>
<td>2476</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>8629</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
</tbody>
</table>

- **N** dense
- 1 sparse
- 1 sparse
- 1 sparse
Tailored Approach — Part 2

Employ a Hybrid Solution

<table>
<thead>
<tr>
<th>Density</th>
<th>Count</th>
<th>N</th>
<th>dense</th>
</tr>
</thead>
<tbody>
<tr>
<td>991</td>
<td>681</td>
<td>1978</td>
<td>· · ·</td>
</tr>
<tr>
<td>0</td>
<td>2476</td>
<td>0</td>
<td>· · ·</td>
</tr>
<tr>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
</tr>
<tr>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
</tr>
<tr>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
</tr>
<tr>
<td>0</td>
<td>8629</td>
<td>0</td>
<td>· · ·</td>
</tr>
</tbody>
</table>
Tailored Approach — Part 2

Employ a Hybrid Solution

Split into *sparse* partitions

<table>
<thead>
<tr>
<th>Density Count</th>
<th>991</th>
<th>681</th>
<th>1978</th>
<th>...</th>
<th>...</th>
<th>517</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2476</td>
<td>0</td>
<td></td>
<td>...</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0</td>
<td>8629</td>
<td>0</td>
<td></td>
<td>...</td>
<td>...</td>
<td>0</td>
</tr>
</tbody>
</table>

**Tailored Approach** — Part 2

Employ a Hybrid Solution

Split into *sparse* partitions

<table>
<thead>
<tr>
<th>Density Count</th>
<th>991</th>
<th>681</th>
<th>1978</th>
<th>...</th>
<th>...</th>
<th>517</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2476</td>
<td>0</td>
<td></td>
<td>...</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0</td>
<td>8629</td>
<td>0</td>
<td></td>
<td>...</td>
<td>...</td>
<td>0</td>
</tr>
</tbody>
</table>
Tailored Approach — Part 2

Employ a Hybrid Solution

Split into sparse partitions

<table>
<thead>
<tr>
<th>$z_0$</th>
<th>$z_1$</th>
<th>$\cdot$</th>
<th>$\cdot$</th>
<th>$\cdot$</th>
<th>$\cdot$</th>
<th>$z_N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>991</td>
<td>681</td>
<td>1978</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>517</td>
</tr>
<tr>
<td>0</td>
<td>2476</td>
<td>0</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>0</td>
</tr>
<tr>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
</tr>
<tr>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
</tr>
<tr>
<td>0</td>
<td>8629</td>
<td>0</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>0</td>
</tr>
</tbody>
</table>

Density Count

N dense
Tailored Approach — Part 2

Employ a Hybrid Solution

<table>
<thead>
<tr>
<th>$z_0$</th>
<th>991</th>
<th>681</th>
<th>1978</th>
<th>$\cdots$</th>
<th>$\cdots$</th>
<th>517</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z_1$</td>
<td>0</td>
<td>2476</td>
<td>0</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>0</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td>$z_N$</td>
<td>0</td>
<td>8629</td>
<td>0</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>0</td>
</tr>
</tbody>
</table>

Split into \textit{sparse} partitions

Hybrid Join

Density Count

$N$ dense

Tailored Approach — Part 2

Employ a Hybrid Solution

<table>
<thead>
<tr>
<th>$z_0$</th>
<th>991</th>
<th>681</th>
<th>1978</th>
<th>$\cdots$</th>
<th>$\cdots$</th>
<th>517</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z_1$</td>
<td>0</td>
<td>2476</td>
<td>0</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>0</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td>$z_N$</td>
<td>0</td>
<td>8629</td>
<td>0</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>0</td>
</tr>
</tbody>
</table>
### Tailored Approach — Part 2

Employ a Hybrid Solution

Each partition has just 1 nonzero computation

<table>
<thead>
<tr>
<th></th>
<th>(991, (z_0))</th>
<th>(681, (z_0))</th>
<th>(1978, (z_0))</th>
<th></th>
<th>(517, (z_0))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, (z_1))</td>
<td>(2476, (z_1))</td>
<td>(0, (z_1))</td>
<td></td>
<td></td>
<td>(0, (z_1))</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0, (z_N))</td>
<td>(8629, (z_N))</td>
<td>(0, (z_N))</td>
<td></td>
<td></td>
<td>(0, (z_N))</td>
</tr>
</tbody>
</table>
Tailored Approach — Part 2

Employ a Hybrid Solution

Each partition has just 1 nonzero computation

<table>
<thead>
<tr>
<th>(991, $z_0$)</th>
<th>(681, $z_0$)</th>
<th>(1978, $z_0$)</th>
<th></th>
<th></th>
<th>(517, $z_0$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, $z_1$)</td>
<td>(2476, $z_1$)</td>
<td>(0, $z_1$)</td>
<td></td>
<td></td>
<td>(0, $z_1$)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0, $z_N$)</td>
<td>(8629, $z_N$)</td>
<td>(0, $z_N$)</td>
<td></td>
<td></td>
<td>(0, $z_N$)</td>
</tr>
</tbody>
</table>
Tailored Approach — Part 2

Employ a Hybrid Solution

Each partition has just 1 nonzero computation

<table>
<thead>
<tr>
<th>991 \cdot z_0</th>
<th>681 \cdot z_0</th>
<th>1978 \cdot z_0</th>
<th>\cdots</th>
<th>\cdots</th>
<th>\cdots</th>
<th>517 \cdot z_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2476 \cdot z_1</td>
<td>0</td>
<td>\cdots</td>
<td>\cdots</td>
<td>\cdots</td>
<td>0</td>
</tr>
<tr>
<td>\cdots</td>
<td>\cdots</td>
<td>\cdots</td>
<td>\cdots</td>
<td>\cdots</td>
<td>\cdots</td>
<td>\cdots</td>
</tr>
<tr>
<td>\cdots</td>
<td>\cdots</td>
<td>\cdots</td>
<td>\cdots</td>
<td>\cdots</td>
<td>\cdots</td>
<td>\cdots</td>
</tr>
<tr>
<td>0</td>
<td>8629 \cdot z_N</td>
<td>0</td>
<td>\cdots</td>
<td>\cdots</td>
<td>\cdots</td>
<td>0</td>
</tr>
</tbody>
</table>
Implementation
Implementation

• Cluster-computing framework on Apache Spark
Implementation

• Cluster-computing framework on Apache Spark

• System written in Java (~10k lines of code)
Implementation

• Cluster-computing framework on Apache Spark

• System written in Java (~10k lines of code)

• Experiments on Amazon EC2:
  • r3.8xlarge instances (32 vCPUs, 244 GiB of memory)
Largest Supported Circuit Size

\[ \text{log}_2 \text{circuit size} \]

# machines

DIZK

libsnark

19 20 21 22 23 24 25 26 27 28 29 30 31
Largest Supported Circuit Size

DIZK

libsnark

# machines

$\log_2$ circuit size

19 20 21 22 23 24 25 26 27 28 29 30 31
Largest Supported Circuit Size

libsnark

~4 million gates

# machines

DIZK

log₂ circuit size
Largest Supported Circuit Size

The chart shows the largest supported circuit size for different numbers of machines. The libsnark benchmark is highlighted with a note indicating it supports around 4 million gates. The chart uses a logarithmic scale to represent the number of gates, with the log₂ circuit size on the x-axis and the number of machines on the y-axis.
Largest Supported Circuit Size

- DIZK: ~2 billion gates
- libsnark: ~4 million gates
Largest Supported Circuit Size

- DIZK
  - ~2 billion gates
- libsnark
  - ~4 million gates

Double # of machines, → twice the circuit size

Double # of machines, → twice the circuit size

~2 billion gates
Scalability

**Distributed Setup**

**Distributed Prover**

---

<table>
<thead>
<tr>
<th># machines</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>256</th>
</tr>
</thead>
</table>
Scalability

Double the circuit size → twice the time
Parallelism

**Distributed Setup**

- Circuit size
  - $2^{15}$ - $2^{16}$
  - $2^{17}$ - $2^{18}$
  - $2^{19}$ - $2^{20}$
  - $2^{21}$ - $2^{22}$
  - $2^{23}$ - $2^{24}$
  - $2^{25}$ - $2^{26}$
  - $2^{27}$ - $2^{28}$
  - $2^{29}$ - $2^{30}$

**Distributed Prover**

- Log$_2$ time (sec)
- # machines
Parallelism

Double # of machines $\rightarrow$ twice as fast

- Circuit size:
  - $2^{15}$ - $2^{16}$
  - $2^{17}$ - $2^{18}$
  - $2^{19}$ - $2^{20}$
  - $2^{21}$ - $2^{22}$
  - $2^{23}$ - $2^{24}$
  - $2^{25}$ - $2^{26}$
  - $2^{27}$ - $2^{28}$
  - $2^{29}$ - $2^{30}$

- Distributed Setup

- Distributed Prover
Conclusion
## Conclusion

<table>
<thead>
<tr>
<th></th>
<th>Prior zkSNARKs</th>
<th>DIZK</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Maximum circuit size</strong></td>
<td>Millions of gates</td>
<td>Billions of gates</td>
</tr>
<tr>
<td><strong>Cost per gate</strong></td>
<td>1ms</td>
<td>10μs</td>
</tr>
</tbody>
</table>
## Conclusion

<table>
<thead>
<tr>
<th></th>
<th>Prior zkSNARKs</th>
<th>DIZK</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Maximum circuit size</strong></td>
<td>Millions of gates</td>
<td>Billions of gates</td>
</tr>
<tr>
<td><strong>Cost per gate</strong></td>
<td>1ms</td>
<td>10µs</td>
</tr>
</tbody>
</table>

- DIZK ([dizk.org](http://dizk.org), open-source, MIT License)
Open Questions
Open Questions

Even Larger Circuits

What techniques will get us to **trillions of gates**, if any?

*(Now, we would need ~100,000 machines in the best case scenario, i.e. too many)*
Open Questions

Even Larger Circuits
What techniques will get us to **trillions of gates**, if any?  
*(Now, we would need ~100,000 machines in the best case scenario, i.e. too many)*

Other Succinct ZKPs
How efficiently can **other succinct ZKPs** be distributed?  
*(STARKs, Bulletproofs, …)*

Our techniques are likely an excellent starting point.