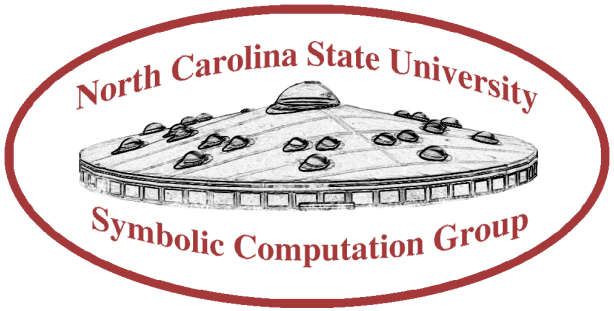
# Black Box Linear Algebra with the LinBox Library

William J. Turner
North Carolina State University
www.math.ncsu.edu/~wjturner



# Overview

1. Black Box Linear Algebra

2. Project LinBox

3. LinBox Objects

4. Black Box Algorithms

5. Black Box Preconditioners

6. Contributions

7. Future Research

# Black Box Linear Algebra

Black box model of a matrix


Perform linear algebra operations, e.g., $A^{-1}b$ (Wiedemann, 1986)

Precondition to reduce matrix problems to computing minimum polynomials

Delayed matrix multiplication requires efficient matrix-vector products

# Project LinBox

Approximately 20 researchers

Canada, France, and U.S.

Algorithms and software for symbolic linear algebra

Particularly black box matrix methods

37,000 lines of C++ code

Online documentation (Doc++)

# LinBox Objects

## Fields

Parameterized with encapsulated element and random element generator types

C++ types allowed to be elements; contain no information of field

Contain methods for element assignment, equality, arithmetic, IO:

| | | |
|---|---|---|
| $x = y$ | : | `F.assign(x,y)` |
| $x == y$ | : | `F.areEqual(x,y)` |
| $x = y + z$ | : | `F.add(x,y,z)` |
| $x = x + y, x+ = y$ | : | `F.addin(x,y)` |
| `cout << x` | : | `F.write(cout,x)` |

# Vectors

Three types of vectors:

**dense vectors:** "normal" vector,
e.g., STL `vector<Element>`

**sparse sequence vectors:** sequence of pairs of indices and nonzero elements,
e.g., STL `list<pair<integer,Element>>`

**sparse associative vectors:** association of indices to nonzero elements,
e.g., STL `map<integer,Element>`

# Black Box Matrices

Templatized by vector class

Only applications to vector allowed:

| | | |
|---|---|---|
| $x = Ay$ | : | `A.apply(x,y), x = A.apply(y)` |
| $x = Ax$ | : | `A.applyin(x)` |
| $x = A^T y$ | : | `A.transposeapply(x,y), x = A.transposeapply(y)` |
| $x = A^T x$ | : | `A.transposeapplyin(x)` |

Retrieve matrix dimensions:

```
A.rowdim()
A.coldim()
```

# Archetypes

Three uses of archetypes:

1. To define the common object interface; i.e., specify what an explicitly designed class must have.

2. To distribute compiled code and prototype library components.

3. To control code bloat.

Used for all but the most basic objects such as floating point numbers and 10 by 10 matrices.

Not a Java interface; an archetype is an explicit class.

# LinBox Field Archetype

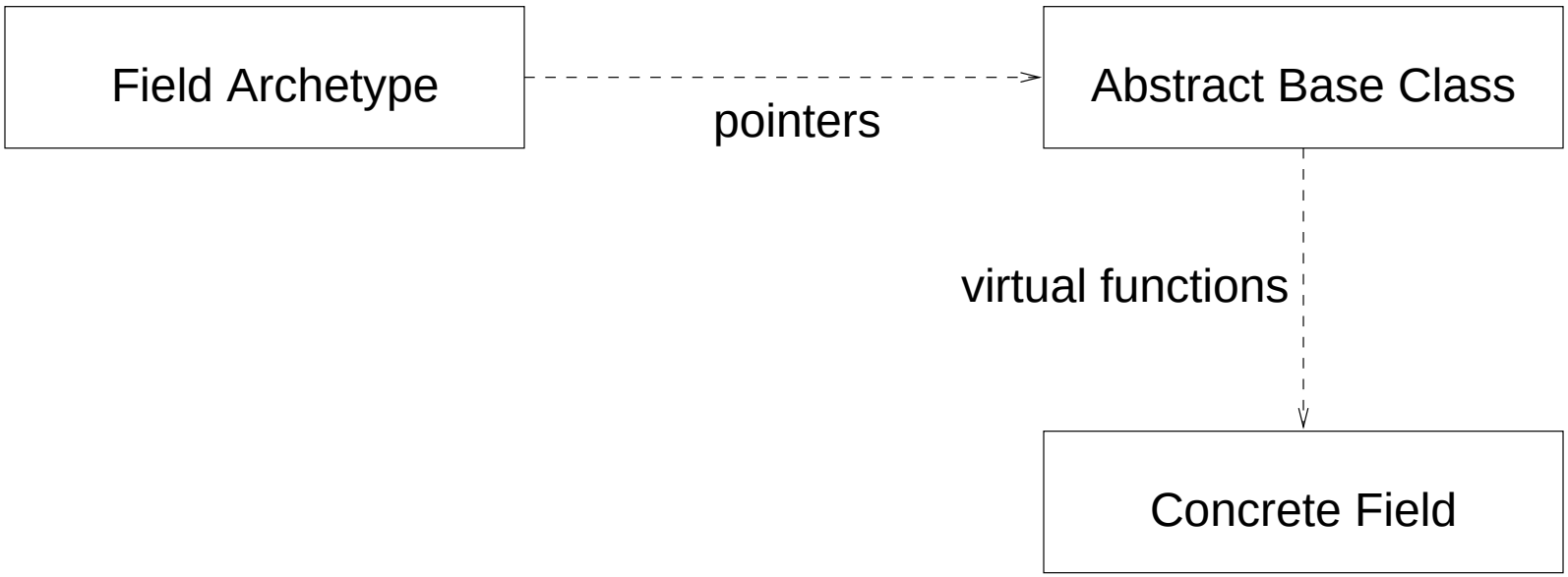
Allow C++ types such as doubles as field elements

$\Downarrow$

Field elements must have (non-virtual) constructors

$\Downarrow$

Field archetype $\neq$ Abstract base class



```
class Field_archetype
{
public:
  // Types
  typedef Element_archetype element;
  typedef RandIter_archetype randIter;

  // Object management
  element& init(element& x, const integer& y = 0 ) const;
   ...

  // Arithmetic
  element& add(element& x, const element& y, const element& z) const;
   ...

  // In-place arithmetic
  element& addin(element& x, const element& y) const;
   ...

  // I/O
  ostream& write(ostream& os) const;
  istream& read(istream& is);
  ostream& write(ostream& os, const element& x) const;
  istream& read(istream& is, element& x) const;
};
```

# Black Box Matrix Archetype

No C++ types to be used as black box matrices

$\Downarrow$

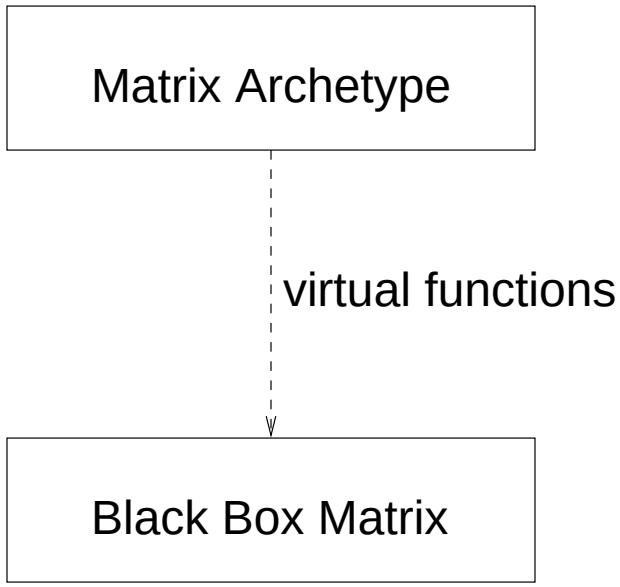
Black box matrices do not need (non-virtual) constructors

$\Downarrow$

Use virtual `clone()` to construct derived classes

$\Downarrow$

Black box matrix archetype $=$ Abstract base class



```
template <class Vector> class Blackbox_archetype
{
public:
  virtual ~Blackbox_archetype() {}
  virtual Blackbox_archetype* clone() const = 0;
  virtual Vector& apply(const Vector& x) const = 0;
  virtual Vector& apply(Vector& y, const Vector& x) const
    { return y = apply(x); }
  virtual Vector& applyin(Vector& x) const
    { Vector y(x); return x = apply(y); }
  virtual Vector& applyTranspose(const Vector& x) const = 0;
  virtual Vector& applyTranspose(Vector& y, const Vector& x) const
    { return y = applyTranspose(x); }
  virtual Vector& applyinTranspose(Vector& x) const
    { Vector y(x); return x = applyTranspose(y); }
  virtual size_t rowdim(void) const = 0;
  virtual size_t coldim(void) const = 0;

protected:

  Blackbox_archetype(void) {}

};
```

# Black Box Algorithms

## Wiedemann Method

For $u, b \in \mathbb{F}^n$ and $A \in \mathbb{F}^{n \times n}$, let

- $f^A =$ minimal polynomial of $A$
- $f^{A,b} =$ minimal recurrence polynomial of $\{A^i b\}_{i=0,1,\ldots}$
- $f_u^{A,b} =$ minimal recurrence polynomial of $\{u^\mathsf{T} A^i b\}_{i=0,1,\ldots}$

Then,

$$f_u^{A,b} \mid f^{A,b} \mid f^A \mid \det(xI - A)$$

Berlekamp-Massey algorithm (1967) computes minimal linear generator of $\{u^\mathsf{T} A^i b\}_{i=0,1,\ldots}$

Let $f_u^{A,b} = x^m + c_{m-1}x^{m-1} + \cdots + c_0$

Wiedemann (1986):

**Linear Solver:** Choose random $u$ so $f_u^{A,b} = f^{A,b}$ and a solution to $Ax = b$ is

$$x = -\frac{1}{c_0}(A^{m-1}b + c_{m-1}A^{m-2}b + \cdots + c_1 b)$$

**Minimal Polynomial:** Choose random $u$ and $b$ so

$$f_u^{A,b} = f^{A,b} = f^A$$

**Determinant:** Randomly perturb $A$ and choose random $u$ and $b$ so

$$f_u^{A,b} = f^{A,b} = f^A = \det(xI - A)$$

and

$$\det(A) = (-1)^n c_0$$

# Baby Steps/Giant Steps Determinant Algorithm

For $A \in \mathbb{F}^{n \times n}$ dense, can compute $a_i = u^{\mathsf{T}} A^i b$ for $i = 0, 1, \ldots, 2n-1$ efficiently by baby steps/giant steps method (Kaltofen & Villard, 2001):

For some $c$, let $r = \lceil \sqrt{cn} \rceil$ and $s = \lceil 2n/r \rceil$

1. (Baby Steps) For $j = 1, 2, \ldots, r-1$ Do $v^{[j]} \leftarrow A^j v$;

2. $Z \leftarrow A^r$;

3. (Giant Steps) For $k = 1, 2, \ldots, s$ Do $(u^{[k]})^{\mathsf{T}} \leftarrow u^{\mathsf{T}} Z^k$;

4. For $k = 0, 1, 2, \ldots, s$ Do
   For $j = 0, 1, 2, \ldots, r-1$ Do $a_{kr+j} \leftarrow \langle u^{[k]}, v^{[j]} \rangle$;

If $A \in \mathbb{Z}^{n \times n}$, compute $\det(A) \bmod p$ for several primes and Chinese remainder to recover $\det(A)$.

# Kaltofen-Saunders Algorithms

Let $A \in \mathbb{F}^{n \times n}$ have rank $r < n$.

Then, Kaltofen & Saunders (1991) gives us:

**Rank:**

Let

- $A$ have leading principle minors nonzero up to $A_r$
- $D$ be a random diagonal matrix

Then,

$$r = \deg(f^{AD}) - 1$$

## Linear Solver:

Let

- $A$ have leading principle minors nonzero up to $A_r$
- $A' \in \mathbb{F}^{r \times n}$ be the matrix formed by first $r$ rows of $A$
- $b \in \mathbb{F}^n$ be such that $Ax = b$ is solvable in $x \in \mathbb{F}^n$
- $x_0 \in \mathbb{F}^n$ be such that $Ax_0 = b$
- $v \in \mathbb{F}^n$ be random

Then,

$$\begin{pmatrix} A_r^{-1} A'(x_0 + v) \\ 0 \end{pmatrix} - v$$

uniformly samples the solution manifold of $Ax = b$

# Preconditioners

Schwartz-Zippel Lemma (Schwartz, 1980; Zippel, 1979):

If

- $f \in \mathbb{F}[x_1, \ldots, x_m]$
- $f \neq 0$
- $n = \deg(f)$
- $S \subseteq \mathbb{F}$, $S$ finite
- $a_1, \ldots, a_m$ selected uniformly and independently from $S$

Then

$$\text{Prob}(f(a_1, \ldots, a_m) \neq 0) \geq 1 - \frac{n}{|S|}$$

# Determinant-Preserving Preconditioner

To compute determinant, want $\tilde{A}$ nonsingular and

$$f^{\tilde{A}} = \det(xI - \tilde{A})$$

**Wiedemann (1986):** Permute or mix columns so all leading principle minors are nonzero and multiply by diagonal matrix

$$\tilde{A} = APD$$

**Chen *et al.* (2001):** Multiply by diagonal matrix

$$\tilde{A} = AD$$

These increase determinant

Baby steps/giant steps algorithm uses more primes for Chinese remaindering

**Theorem 1.** *Let $\mathbb{F}$ be a field, $A \in \mathbb{F}^{n\times n}$ be nonsingular, and $S$ be a finite subset of $\mathbb{F}$. If*

$$U = \begin{bmatrix} 1 & a_1 & & \\ & \ddots & \ddots & \\ & & 1 & a_{n-1} \\ & & & 1 \end{bmatrix}$$

*where $a_1, \ldots, a_{n-1}$ are chosen uniformly and independently from $S$, then $AU$ is nonsingular and*

$$f^{AU} = \det(xI - AU)$$

*with probability at least $1 - n(n-1)/(2\,|S|)$.*

Can also use $AU^{\mathsf{T}}$, $UA$, and $U^{\mathsf{T}}A$.

## Idea behind proof:

$$\mathcal{U} = \begin{bmatrix} 1 & \alpha_1 & & \\ & \ddots & \ddots & \\ & & 1 & \alpha_{n-1} \\ & & & 1 \end{bmatrix}$$

Smith form of $xI - A\mathcal{U}$ is

$$\begin{bmatrix} 1 & & & \\ & \ddots & & \\ & & 1 & \\ & & & f^{A\mathcal{U}} \end{bmatrix}$$

There exists $y$ such that

$$y, (A\mathcal{U})y, \ldots, (A\mathcal{U})^{n-1}y$$

are linearly independent.

The determinant of the matrix with these columns is a nonzero polynomial of degree no more than $n(n-1)/2$ in $\alpha_1, \ldots, \alpha_{n-1}$.

Apply Schwartz-Zippel to find probability that determinant of matrix with columns

$$y, (AU)y, \ldots, (AU)^{n-1}y$$

is nonzero.

# Butterfly Network Preconditioner

For Kaltofen-Saunders linear solver, need to permute or mix rows and columns so leading principle minor $A_r$ nonzero

**Wiedemann (1986):** Extend $A$ so $n = 2^k$ and use parameterized matrix based on Beneš network

$$\tilde{A} = AP$$

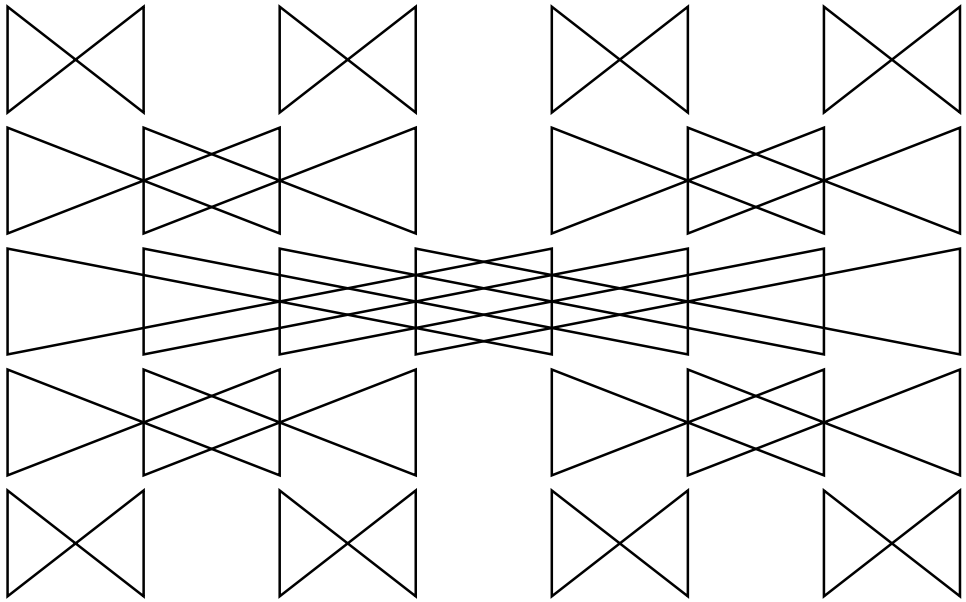
**Kaltofen & Saunders (1991):** Pre- and postmultiply by random unit upper and lower triangular Toeplitz matrices

$$\tilde{A} = T_1 A T_2$$

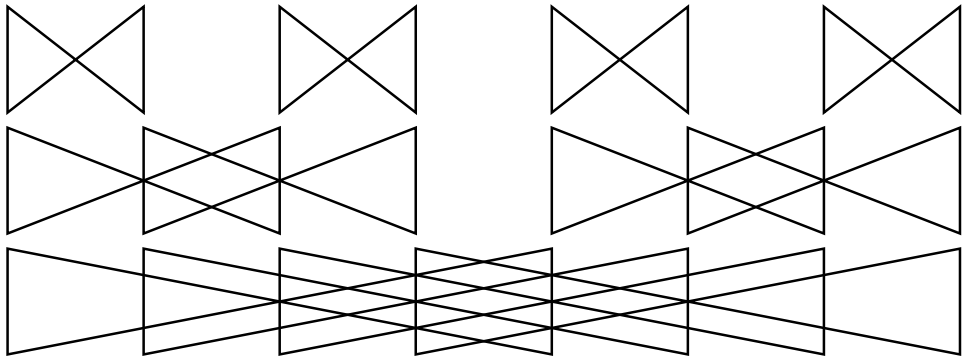
**Chen *et al.* (2001):** Do not extend $A$ and use two butterfly networks

$$\tilde{A} = B_1 A B_2$$

Beneš network:



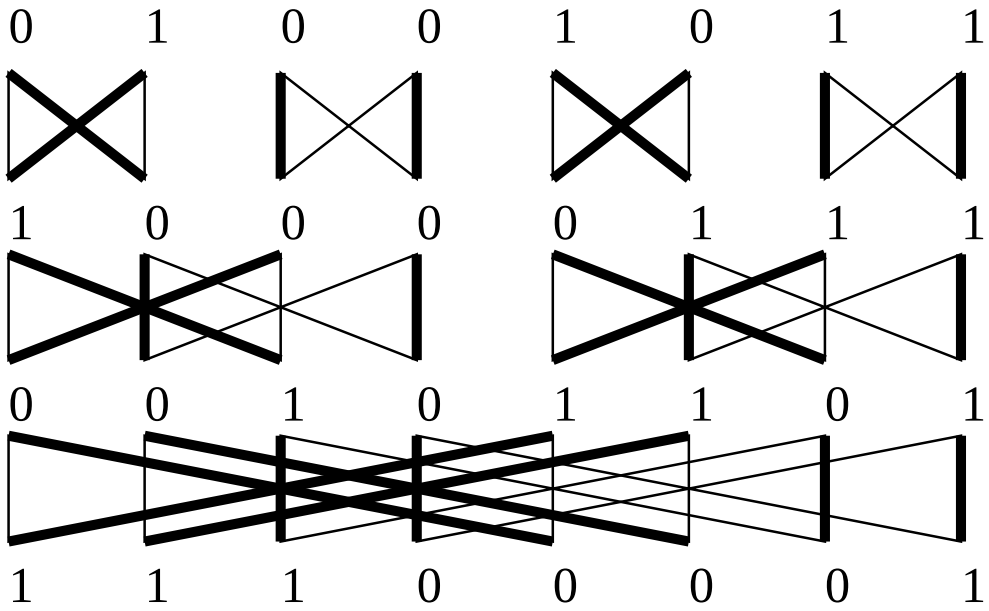
Butterfly network:



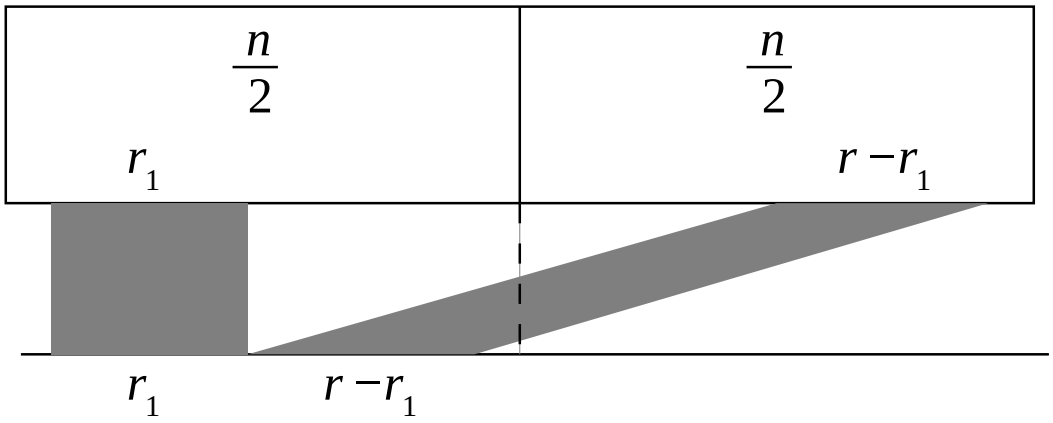
**Lemma 1.** *Let $n = 2^l$. The $l$-dimensional butterfly network discussed above can switch any $r$ indices*

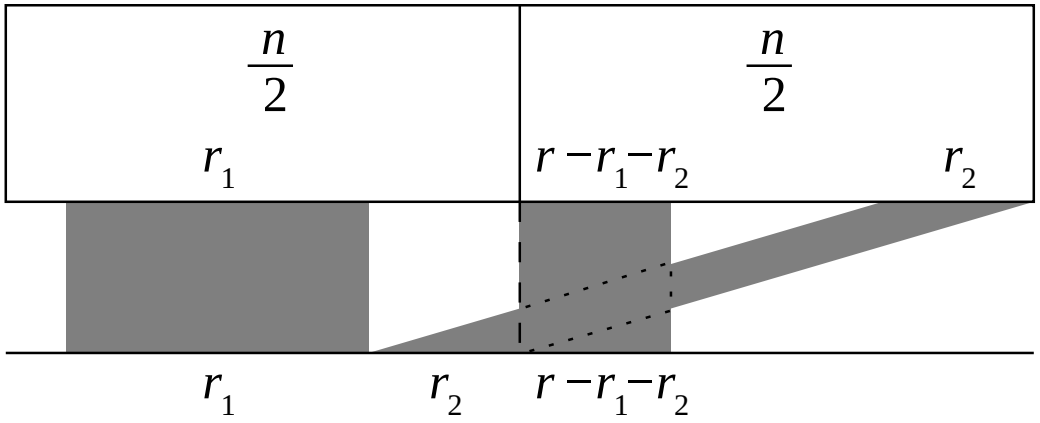
$$1 \le i_1 < \cdots < i_r \le n$$

*into any desired contiguous block of indices; wrap around outside, for our purposes, shall preserve contiguity. Furthermore, the network contains a total of $n \log_2(n)/2$ switches.*



# Idea behind proof:

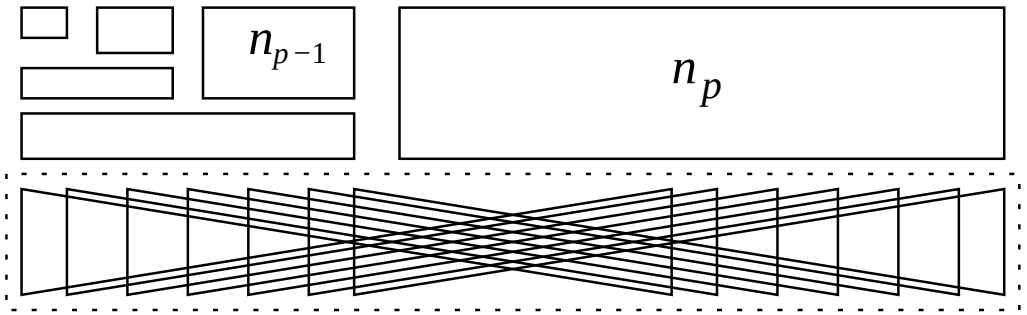


If $n \neq 2^l$,

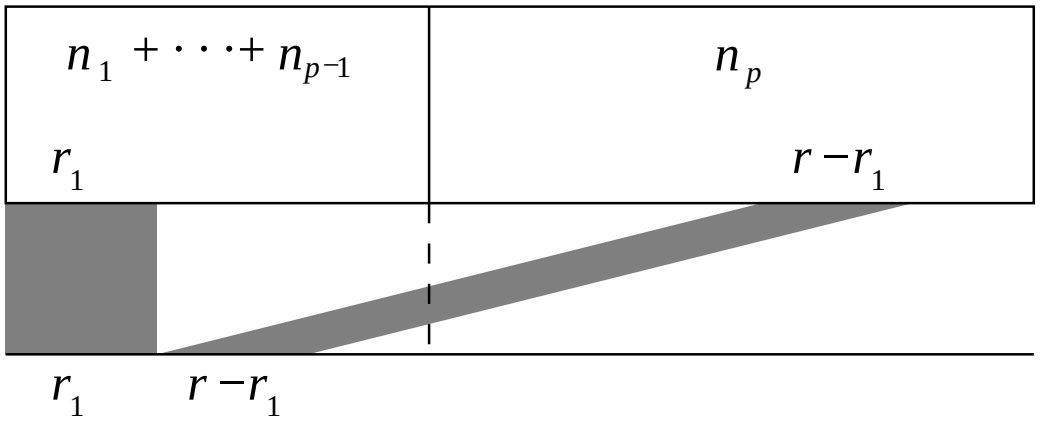
$$n = \sum_{i=1}^{k} 2^{l_i} \text{ where } l_1 < l_2 < \cdots < l_p; \text{ let } n_i = 2^{l_i}$$

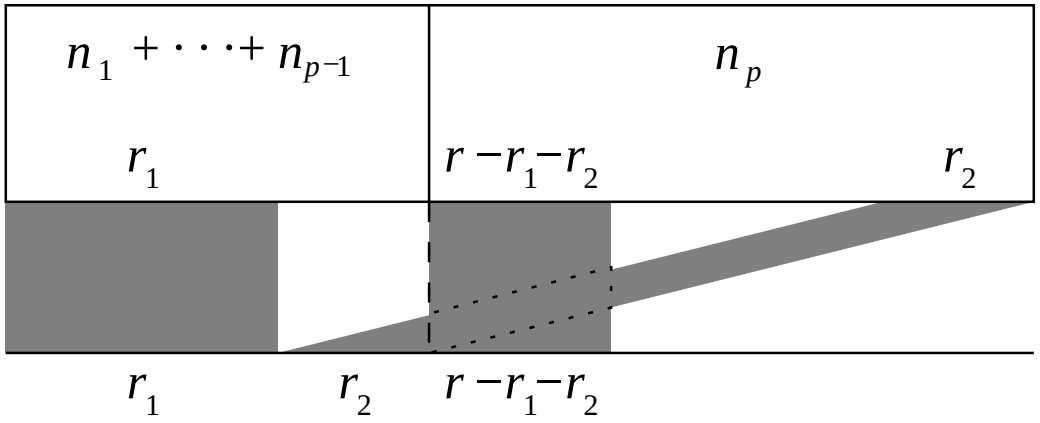
1. Lay out butterfly network for each $n_i$ block
2. Connecting by butterfly switches recursively such that $\sum_{i=1}^{k-1} n_i$ merged with far right nodes of $n_k$



**Theorem 2.** *The generalized butterfly network discussed above can switch any* $r$ *indices* $1 \leq i_1 < \cdots < i_r \leq n$ *into the contiguous block* $1, 2, \ldots, r$. *Furthermore, it has a depth of* $\lceil \log_2(n) \rceil$ *and a total of no more than* $n\lceil \log_2(n) \rceil / 2$ *butterfly switches.*

# Idea behind proof:



# Contributions

- 22,000 lines of C++ code
  - Archetypes
  - Sparse associative vectors
  - Examples: Fields, Black Box matrices, Vectors
  - Wiedemann determinant algorithm
  - Tests
- Online (Doc++) documentation
- Preconditioner proofs based on Smith form
- Butterfly network proof

# Future Research

I plan to do at least one of the following:

**Black Box Rank Algorithm:** Investigate using a block Wiedemann method to compute the rank of a black box matrix

**Block Lanczos:** Implement a block Lanczos linear solver in LinBox

**Smith Form Preconditioners:** Further investigate applying properties of proofs to other problems

# References


E. R. BERLEKAMP (1968). *Algebraic Coding Theory*. McGraw-Hill, New York.

L. CHEN, W. EBERLY, E. KALTOFEN, B. D. SAUNDERS, W. J. TURNER, & G. VILLARD (2001). Efficient Matrix Preconditioners for Black Box Linear Algebra. *Linear Algebra and Applications*. Special issue on *Infinite Systems of Linear Equations Finitely Specified*.

E. KALTOFEN & B. D. SAUNDERS (1991). On Wiedemann's Method of Solving Sparse Linear Systems. In H. F. MATTSON, T. MORA, & T. R. N. RAO (eds.), *Proc. AAECC-9*, vol. 539 of *Lect. Notes Comput. Sci.*, 29–38. Springer Verlag, Heidelberg, Germany.

E. KALTOFEN & G. VILLARD (2001). On the complexity of computing determinants. In *Proc. Fifth Asian Symposium on Computer Mathematics ASCM 2001*. World Scientific Publishing Company. Invited contribution; extended abstract.

J. L. MASSEY (1969). Shift-Register Synthesis and BCH Decoding. *IEEE Trans. Inf. Theory*, **IT-15**: 122–127.

J. T. SCHWARTZ (1980). Fast Probabilistic Algorithms for Verification of Polynomial Identities. *J. ACM*, **27**: 701–717.

D. H. WIEDEMANN (1986). Solving Sparse Linear Equations Over Finite Fields. *IEEE Trans. Inf. Theory*, **IT-32**: 54–62.

R. ZIPPEL (1979). Probabilistic Algorithms for Sparse Polynomials. In *Proc. EUROSAM '79*, vol. 72 of *Lect. Notes Comput. Sci.*, 216–226. Springer Verlag, Heidelberg, Germany.