$\{0,1\}$-Matrices: The Four Russians and the Mailman

David Saunders
(Univ. of Delaware)

{0,1}-Matrices: The Four Russians and the Mailman

David Saunders
(Univ. of Delaware)

I am a proponent of LinBox,FFlas/FFpack, Givaro
C++ template libraries for exact linear algebra
Google: "Linbox team" to reach github project

ZO and ZOMO

$\{0,1\}$- and $\{0,1,-1\}$-matrices are ubiquitous.

- Graph adjacency matrix is ZO.
- Graph Laplacian is ZO + D.
- Boundary matrices of simplicial complex are ZOMO.
- Any matrix over GF2 is ZO, over GF3 is ZOMO.
- Many relations are expressed as ZO incidence matrices.
- ZO + very sparse is also seen in practice.
- Block Wiedemann gives opening to use ZO or ZOMO as projectors.

Matrix Multiplication

$$C = AB$$

$$\left(m \times p\right) = \left(m \times n\right) * \left(n \times p\right)$$

Using indices i, j, k in the dimensions m, n, p, respectively.

Definition: of matrix multiplication is that the i, j entry of C is the dot product of the i-th row of A times the j-th column of B.

Matrix Multiplication

$$C = AB$$

$$(m \times p) = (m \times n) * (n \times p)$$

Using indices i, j, k in the dimensions m, n, p, respectively.

Definition: of matrix multiplication is that the i, j entry of C is the dot product of the i-th row of A times the j-th column of B. In the standard three nested loop presentation this is

for i in $[1..m]$
  for k in $[1..p]$
    for j in $[1..n]$
      $c_{i,k} = c_{i,k} + a_{i,j} b_{j,k}$.

Square Matrix Multiplication

- Matrix multiplication costs $O(n^3)$, classical.
- Matrix multiplication costs $O(n^{2.81})$, Strassen.
- Matrix multiplication costs $O(n^{2.38})$, in theory.
- Matrix multiplication costs $O(n^3/\lg(n))$ over GF2, method of 4 Russians.

Square matrix multiplication

BLAS gemm is really fast. How fast?

n	naive	blas	speedup
50	8.1e-05	6e-06	13.5
100	0.000848	3.2e-05	26.5
500	0.124	0.0025	49.6
1000	1.002	0.018	55.7
5000	174.156	2.04	85.4

How is it done?

Square matrix multiplication

BLAS gemm is really fast. How fast?

n	naive	blas	speedup
50	8.1e-05	6e-06	13.5
100	0.000848	3.2e-05	26.5
500	0.124	0.0025	49.6
1000	1.002	0.018	55.7
5000	174.156	2.04	85.4

How is it done?

Not by reduction in number of field operations but by attention to hardware (caches, pipelines, simd instructions, etc.)

Block Wiedemann algorithm

Matrix A is $n \times n$, $b \ll n$

- $n \times b$: $V_i = A^i V$, right projection
- $b \times b$: $S_i = UA^i V$, left projection
- $S_i \longrightarrow$ SigmaBasis $\longrightarrow$ MatrixMinpoly
- MatrixMinpoly $\longrightarrow$(whp) leading Frobenius invariants, particularly minpoly, perhaps charpoly.
- minpoly $\longrightarrow$ solve nonsingular,
  charpoly $\longrightarrow$ determinant
  (perhaps) leading invariants $\longrightarrow$ rank, nullspace,

Block Wiedemann algorithm

Matrix A is $n \times n$, $b \ll n$

- $n \times b$: $V_i = A^i V$, right projection
- $b \times b$: $S_i = UA^i V$, left projection
- $S_i \longrightarrow$ SigmaBasis $\longrightarrow$ MatrixMinpoly
- MatrixMinpoly $\longrightarrow$(whp) leading Frobenius invariants, particularly minpoly, perhaps charpoly.
- minpoly $\longrightarrow$ solve nonsingular,
  charpoly $\longrightarrow$ determinant
  (perhaps) leading invariants $\longrightarrow$ rank, nullspace,

Block Wiedemann dominant steps

Matrix A is $n \times n$, $b \ll n$
$V_0 = V$ is $n \times b$, random. U is $b \times n$.
Wiedemann: $b = 1$, repeat $2n$ times:

1. $V_i = AV_{i-1}$
2. $s_i = UV_i$, s_i are scalars.

Block Wiedemann dominant steps

Matrix A is $n \times n$, $b \ll n$
$V_0 = V$ is $n \times b$, random. U is $b \times n$.
Wiedemann: $b = 1$, repeat $2n$ times:

1. $V_i = AV_{i-1}$
2. $s_i = UV_i$, s_i are scalars.

Coppersmith: repeat about $2n/b$ times:

1. $V_i = AV_{i-1}$
2. $S_i = UV_i$, S_i are $b \times b$.

(same number of A $\times$ column vector in steps 1.)

Block Wiedemann dominant steps

Matrix A is $n \times n$, $b \ll n$
$V_0 = V$ is $n \times b$, random. U is $b \times n$.
Wiedemann: $b = 1$, repeat $2n$ times:

1. $V_i = AV_{i-1}$
2. $s_i = UV_i$, s_i are scalars.

Coppersmith: repeat about $2n/b$ times:

1. $V_i = AV_{i-1}$
2. $S_i = UV_i$, S_i are $b \times b$.

(same number of A $\times$ column vector in steps 1. And simd instruction parallelism available!)

Block Wiedemann dominant steps

Matrix A is $n \times n$, $b \ll n$
$V_0 = V$ is $n \times b$, random. U is $b \times n$.
Wiedemann: $b = 1$, repeat $2n$ times:

1. $V_i = AV_{i-1}$
2. $s_i = UV_i$, s_i are scalars.

Coppersmith: repeat about $2n/b$ times:

1. $V_i = AV_{i-1}$
2. $S_i = UV_i$, S_i are $b \times b$.

(same number of A $\times$ column vector in steps 1. And simd instruction parallelism available!)
Wait, step 2 costs more.

Block Wiedemann dominant steps

Matrix A is $n \times n$, $b \ll n$
$V_0 = V$ is $n \times b$, random. U is $b \times n$.
Wiedemann: $b = 1$, repeat $2n$ times:

1. $V_i = AV_{i-1}$
2. $s_i = UV_i$, s_i are scalars.

Coppersmith: repeat about $2n/b$ times:

1. $V_i = AV_{i-1}$
2. $S_i = UV_i$, S_i are $b \times b$.

(same number of A $\times$ column vector in steps 1. And simd instruction parallelism available!)
Wait, step 2 costs more. Does it have to?

Block Wiedemann dominant steps

Matrix A is $n \times n$, $b \ll n$
$V_0 = V$ is $n \times b$, random. U is $b \times n$.
Wiedemann: $b = 1$, repeat $2n$ times:

1. $V_i = AV_{i-1}$
2. $s_i = UV_i$, s_i are scalars.

Coppersmith: repeat about $2n/b$ times:

1. $V_i = AV_{i-1}$
2. $S_i = UV_i$, S_i are $b \times b$.

(same number of A $\times$ column vector in steps 1. And simd instruction parallelism available!)
Wait, step 2 costs more. Does it have to? Make it $\{0,1\}$ or even $(I_b, I_b, \ldots, I_b)$.

focus on row operations in B and C

The order of the loops may be changed and a useful form is when the inner loop is ranging across a rows of B and C:

for i in [1..m]
  for j in [1..n]
    for k in [1..p]
      $c_{i,k} = c_{i,k} + a_{i,j} b_{j,k}.$

focus on row operations in B and C

The order of the loops may be changed and a useful form is when the inner loop is ranging across a rows of B and C:

for i in [1..m]
  for j in [1..n]
    for k in [1..p]
      $c_{i,k} = c_{i,k} + a_{i,j} b_{j,k}$.

If A is a $\{0,1\}$-matrix, the inner loop is row addition.

for i in [1..m]
  for j in [1..n]
    if $a_{i,j} = 1$ then $C_i = C_i + B_j$.

Two ways to focus on row operations

for j in [1..n]
  for i in [1..m]
    if $a_{i,j} = 1$ then $C_i = C_i + B_j$.

or

for i in [1..m]
  for j in [1..n]
    if $a_{i,j} = 1$ then $C_i = C_i + B_j$.

Two methods to speed up multiplication by $\{0,1\}$ or $\{0,1,-1\}$ matrix.

V. Arlazarov, E. Dinic, M. Kronrod, I. Faradev, "On economical construction of the transitive closure of a directed graph", Dokl. Akad. Nauk SSSR, 194 (11). Original in Russian in Dokl. Akad. Nauk SSSR, 134 (3), 1970.

E. Liberty and S. W. Zucker, "The Mailman algorithm: A note on matrix - vector multiplication", Information Processing Letters, Volume 109 (3) 2009.

They have dual structures and complementary strengths vis a vis matrix shape.

4 Russians: look at column(s) of A

$$\begin{pmatrix} \\ C_2 += B_j \\ C_3 += B_j \\ \\ \end{pmatrix} = \begin{pmatrix} & 0 & \\ & 1 & \\ & 1 & \\ & 0 & \end{pmatrix} \times \begin{pmatrix} \\ \\ B_j \\ \\ \end{pmatrix}$$

4 Russians: look at column(s) of A

$$\begin{pmatrix} \\ C_2 += B_j \\ C_3 += B_j \\ \\ \end{pmatrix} = \begin{pmatrix} & 0 & \\ & 1 & \\ & 1 & \\ & 0 & \end{pmatrix} \times \begin{pmatrix} \\ \\ B_j \\ \\ \end{pmatrix}$$

$$\begin{pmatrix} C_1 += B_{j+1} \\ C_2 += (B_j + B_{j+1}) \\ C_3 += B_j \\ C_4 += (B_j + B_{j+1}) \\ \\ \end{pmatrix} = \begin{pmatrix} & 01 & \\ & 11 & \\ & 10 & \\ & 11 & \\ & 00 & \end{pmatrix} \times \begin{pmatrix} \\ \\ B_j \\ B_{j+1} \\ \\ \end{pmatrix}$$

4 Russians: look at column(s) of A

$$\begin{pmatrix} \\ C_2 += B_j \\ C_3 += B_j \\ \\ \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} \times \begin{pmatrix} \\ \\ B_j \\ \\ \end{pmatrix}$$

$$\begin{pmatrix} C_1 += B_{j+1} \\ C_2 += (B_j + B_{j+1}) \\ C_3 += B_j \\ C_4 += (B_j + B_{j+1}) \\ \\ \end{pmatrix} = \begin{pmatrix} 01 \\ 11 \\ 10 \\ 11 \\ 00 \end{pmatrix} \times \begin{pmatrix} \\ \\ B_j \\ B_{j+1} \\ \\ \end{pmatrix}$$

$m + 1$ row adds instead of $2m$ row adds.

Mailman: look at row(s) of A

$$\begin{pmatrix} \\ \\ \\ C_4 = B_2 + B_3 \\ \\ \end{pmatrix} = \begin{pmatrix} & & & & \\ & & & & \\ & & & & \\ 0 & 1 & 1 & 0 & 0 \\ & & & & \end{pmatrix} \times \begin{pmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \end{pmatrix}$$

Mailman: look at row(s) of A

$$\begin{pmatrix} \\ \\ \\ C_4 = B_2 + B_3 \\ \\ \end{pmatrix} = \begin{pmatrix} & & & & \\ & & & & \\ & & & & \\ 0 & 1 & 1 & 0 & 0 \\ & & & & \end{pmatrix} \times \begin{pmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \end{pmatrix}$$

$$\begin{pmatrix} \\ \\ \\ C_4 = B_1 + (B_2 + B_5) \\ C_5 = B_3 + (B_2 + B_5) \end{pmatrix} = \begin{pmatrix} & & & & \\ & & & & \\ & & & & \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \end{pmatrix}$$

Mailman: look at row(s) of A

$$\begin{pmatrix} \\ \\ C_4 = B_2 + B_3 \\ \\ \end{pmatrix} = \begin{pmatrix} & & & & \\ & & & & \\ 0 & 1 & 1 & 0 & 0 \\ & & & & \\ \end{pmatrix} \times \begin{pmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \end{pmatrix}$$

$$\begin{pmatrix} \\ \\ C_4 = B_1 + (B_2 + B_5) \\ C_5 = B_3 + (B_2 + B_5) \\ \\ \end{pmatrix} = \begin{pmatrix} & & & & \\ & & & & \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ & & & & \\ \end{pmatrix} \times \begin{pmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \end{pmatrix}$$

$m + 2$ row adds instead of $2m$ row adds.

Back to four Russians

$$\begin{pmatrix} \\ C_2 {+}{=} B_j \\ C_3 {+}{=} B_j \\ \\ \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} \times \begin{pmatrix} \\ \\ B_j \\ \\ \end{pmatrix}$$

$$\begin{pmatrix} C_1 {+}{=} B_{j+1} \\ C_2 {+}{=} (B_j + B_{j+1}) \\ C_3 {+}{=} B_j \\ C_4 {+}{=} (B_j + B_{j+1}) \\ \\ \end{pmatrix} = \begin{pmatrix} 01 \\ 11 \\ 10 \\ 11 \\ 00 \end{pmatrix} \times \begin{pmatrix} \\ \\ B_j \\ B_{j+1} \\ \\ \end{pmatrix}$$

t columns

Build table of 2^t B-row sums.

$$\begin{aligned} &\dots \\ T_{101} &= T_{001} + B_3 = B_1 + B_3 \\ T_{110} &= T_{010} + B_3 = B_2 + B_3 \\ T_{111} &= T_{011} + B_3 = B_1 + B_2 + B_3 \end{aligned}$$

Using table, sweep down col panel of A to update C row by row.

$$\begin{pmatrix} C_1 {+}{=} T_{110} \\ C_2 {+}{=} T_{010} \\ C_3 {+}{=} T_{101} \\ C_4 {+}{=} T_{110} \\ C_5 {+}{=} T_{011} \\ \dots \end{pmatrix} = \begin{pmatrix} 110 & & \\ 010 & & \\ 101 & & \\ 110 & & \\ 011 & & \\ \dots & & \end{pmatrix} \times \begin{pmatrix} B_1 \\ B_2 \\ B_3 \\ \\ \\ \\ \end{pmatrix}$$

Four Russians analysis

A is a $m \times n$ zero-one matrix.
Panel width is t.
The following two steps must be done n/t times:

1. Table construction, costing 2^t row additions ($2^t - t - 1$ to be precise).
2. Use table to put row combinations into C, costing m row adds.

$$\text{total cost in row additions is } mn/t + n2^t/t).$$

Back to Mailman

$$\begin{pmatrix} \\ \\ \\ C_4 = B_2 + B_3 \\ \\ \end{pmatrix} = \begin{pmatrix} \\ \\ \\ 0 & 1 & 1 & 0 & 0 \\ \\ \end{pmatrix} \times \begin{pmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \end{pmatrix}$$

$$\begin{pmatrix} \\ \\ \\ C_4 = B_1 + B_2 + B_5 \\ C_5 = B_3 + B_2 + B_5 \\ \\ \end{pmatrix} = \begin{pmatrix} \\ \\ \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ \\ \end{pmatrix} \times \begin{pmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \end{pmatrix}$$

t rows

Build table of 2^t B-row sums. Each row of B goes in exactly one sum, indexed by the pattern of C rows to which it contributes.. For instance, with $t = 3$, T_{101} includes B_j when B_j contributes to C_1 and C_3, but not C_2. Next, for each C_i, combine the entries of T that are sums that contribute to C_i (all those T entries for indices with i-th bit on.)

$$\begin{pmatrix} T_{001} + T_{011} + T_{101} + T_{111} \\ T_{010} + T_{011} + T_{110} + T_{111} \\ T_{100} + T_{101} + T_{110} + T_{111} \\ \\ \\ \\ \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ \\ \\ \\ \end{pmatrix} \times \begin{pmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \end{pmatrix}$$

Table handling

$T[000]$
$T[001]$
$T[010]$
$T[011]$
$T[100]$
$T[101]$
$T[110]$
$T[111]$
Add last 4 entries to C_3.

Table handling

$T[000]$
$T[001]$
$T[010]$
$T[011]$
$T[100]$
$T[101]$
$T[110]$
$T[111]$
Add last 4 entries to C_3. Also add them to the first four entries.
$T[*00] = T[000] + T[100]$
$T[*01] = T[001] + T[101]$
$T[*10] = T[010] + T[110]$
$T[*11] = T[011] + T[111]$

Mailman analysis

A is a $m \times n$ zero-one matrix.
Panel width is t.
The following two steps must be done m/t times:

1. Build table using n row additions.
2. Use table to row combinations into C at cost 2×2^t row ops.

$$\text{total cost in row ops is } mn/t + 2m2^t/t).$$

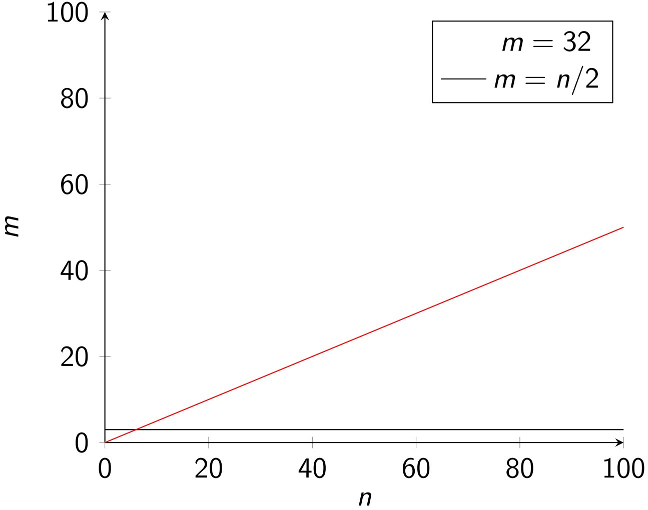
Mailman analysis

A is a $m \times n$ zero-one matrix.
Panel width is t.
The following two steps must be done m/t times:

1. Build table using n row additions.
2. Use table to row combinations into C at cost 2×2^t row ops.

total cost in row ops is $mn/t + 2m2^t/t)$.
Compare 4 Russians: $mn/t + n2^t/t$.

Map for choosing method



Timing over Z_{10003}

A = time of 8×80000 ZO matrix times 80000×1000 matrix

vs

B = time of 8 reps of 1×80000 dense vector times 80000×1000 matrix.

11-fold speedup $B/A = 11$.

$C = B$ with ZO vector, speedup $C/A \approx 7$.

Example: Solve nonsingular system

1. Minpoly via Block Wiedemann using $U \in ZO^{4 \times n}$ and rational (poly) linear system solve with random rhs.
   m(x) $= \sum_{i=0}^{d} m_i x^i$. [$2d$ mv's]
2. $x = (-1/m_0 \sum_{i=1}^{d} m_i A^{i-1} b$. [$d - 1$ mv's]
3. Check $Ax = b$ [1 mv]. Go to 1 if fail, else return x.

Example: Solve nonsingular system

1. Minpoly via Block Wiedemann using $U \in ZO^{4\times n}$ and rational (poly) linear system solve with random rhs.
   m(x) $= \sum_{i=0}^{d} m_i x^i$. [$2d$ mv's]
2. $x = (-1/m_0 \sum_{i=1}^{d} m_i A^{i-1} b$. [$d-1$ mv's]
3. Check $Ax = b$ [1 mv]. Go to 1 if fail, else return x.

Block Wiedemann is faster than $b = 1$ Wiedemann because of simd in mv's and Mailman in panel products and tiny block size. Probability of success is adequate. Expected number of repetitions is $1 + \epsilon$. C

Method duality

	4 Russians	Mailman
matrix use in kernel		
A	$m \times t$ (few cols) each row a t bit index	$t \times n$ (few rows) each col a t bit index
C	update all rows	write t rows, done with
B	read t rows, done with	reread all rows
The table's two phases		
build it use it	$B \longrightarrow T$, indep of A scan A building is overhead	scan A $T \longrightarrow C$, indep of A using is overhead