Faster Boolean Matrix Operations, including Extension Fields

Gregory V. Bard
June 21, 2006.

Part One: What is so hard about Char 2?

- Matrix Operations over finite fields (particularly $GF(2)$ are used in)
  - Stream Cipher Cryptanalysis
  - Integer Factorization
  - Attacks on Multivariate Polynomial Signature Schemes
  - Error Correcting Codes
  - Graph Theory (semiring)
- ...and linear algebra has been around for a *very long* time, with *many* books and articles.
- Yet, many questions about finite field matrix operaitons are unanswered, and naive algorithms are often used in practice, even if performance is critical.

The Fields Cannot be Ordered

- No finite field, or even a field of non-zero characteristic, can be ordered.
- An ordering of a field can be thought of either through "the positivity axioms" or "a total ordering that respects the addition operation."
- By "total ordering" we mean that for all a, b, one of three things is true: $a > b$, $a = b$, or $a < c$, and also transitivity and anti-symmetry.
- By "respects the operation" we mean that

$$\text{if } a > b \text{ and } c > d \text{ then } a + c > b + d$$

- Of course this means if $a > 0$ and $c > 0$ then $a + c > 0$, or that the set of elements greater than zero is closed under the addition operation.
- Since $0 \neq 1$ either $1 > 0$ or $1 < 0$. Pick one, and consider

$$1 + 1; 1 + 1 + 1; 1 + 1 + 1 + 1; \cdots$$

Some Concepts Break

- I don't have time to show all this now, but ask me afterward if you like...
- The "norm induced by the dot product" is not a norm, but a seminorm. (i.e. non-zero vectors $\vec{x}$ can still have $|\vec{x}| = 0$). Consider

$$|(\underbrace{1, 1, 1, \ldots, 1}_{p \text{ times}}, 0, 0, \ldots, 0)|$$

- This is because the dot product is not a positive-definite operator when considered as an element of the dual space.
- One can show that all non-trivial seminorms are not norms.
- Since the ring of matrices over a field is vector space over that field, this means there are no matrix norms either.

Some Algorithms Break

- Gram-Schmidt fails on almost any random example in $GF(2)$.
- ...this is because the dot-product is not a positive-definite operator in the space of linear functionals.
- The Cholesky Factorization is one where $A = LL^T$, and so $L_{11} = \sqrt{A_{11}}$.
- Square-roots are not available for all elements in fields of characteristic not equal to two. (Otherwise since the multiplicative group of $GF(q)$ has order $q - 1$):

$$(x^{q/2})^2 = x^q = x^{q-1}x = x$$

- For fields of characteristic two, I have modified version of Cholesky that is fast, but fails in practice. (More on that later).
- The series $Ax, A^2x, A^3x, \ldots$ does not eventually produce an eigenvector. This is because there is no concept of convergence when the set is finite. (Only sets with the constant sequence as a suffix will converge).

Probability of Invertibility

- Amazingly, the probability of a random matrix being singular is nonzero, and can be calculated explicitly!
- A matrix is invertible if and only if it is of full rank, that is that its columns form a basis of the space.
- How many such matrices can I build?
  - Recall an n-dimensional sub-space has q^n elements in it.
  - The first column can be any non-zero vector. ($q^n - 1$ choices).
  - The second column cannot be a scalar multiple of the first. ($q^n - q$ choices).
  - The third column cannot be in the subspace generated by the first two. ($q^n - q^2$ choices).
  - The ith thus can be one from $q^n - q^{i-1}$ choices.
  - So I can build $\Pi_{i=1}^{i=n} \left(q^n - q^{i-1}\right)$ such matrices.

Probability of a Random Matrix Being Invertible

- Knowing that I can build $\prod_{i=1}^{i=n} \left(q^n - q^{i-1}\right)$ such matrices.
- There are q^{n^2} matrices of size $n \times n$. Thus the probability of a random matrix being non-singular is the ratio of these two numbers.

$$\prod_{i=1}^{i=n} \left(1 - q^{i-1-n}\right)$$

- $GF(2)$ is 0.2888.
- $GF(16)$ is 0.9336.
- $GF(256)$ is 0.9961.
- $GF(2^{32})$ is $1 - 2.328 \times 10^{-9}$.
- Rational Arithmetic... essentially one.

Complexity Calculations

- In "normal" linear algebra over $\mathbb{Q}, \mathbb{R}, \mathbb{C}$ one can count "floating point operations." but... there are no floating point operations in finite fields!
- To count all operations would require tracking data addressing methods, loop iterators, cache coherency, branch prediction, etc...
- Therefore I propose counting matrix memory reads and writes.
- Field operations are a single gate for $GF(2)$, or a table lookup for $GF(256)$, whereas reads and writes require memory transactions. With multiple execution queues, dual cores, and vector processing, bulk rates are quite fast (128 bits/clock cycle?)
- The matrices are too big for the cache. All data must get from the memory to the processor and back, so bus timing is the critical metric. Throughputs can be upto 1/2 Gigabyte/sec, or about 4–8 bits per clock cycle.
- Finally, coefficients count, so instead of Big-Oh, I use

$$f(x) \sim g(x) \iff \lim_{x \to \infty} \frac{f(x)}{g(x)} = 1$$

Adjusting Sizes

- For simplicity I will assume matrix dimensions are divisible by various numbers. If not, note that

$$\begin{bmatrix} A & 0 \\ 0 & I \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1} & 0 \\ 0 & I \end{bmatrix}$$

- and also

$$\begin{bmatrix} A & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} B & 0 \\ 0 & I \end{bmatrix} = \begin{bmatrix} AB & 0 \\ 0 & I \end{bmatrix}$$

Part Two: Complexity of Linear Algebra (Review)

- Basic Operations
  - Matrix Multiplication
  - Matrix Inversion
  - LU-Factorization
  - QR-Factorization
- A blackbox algorithm for one of these provides an algorithm of equal complexity for all the others.
- Until 1969 it was thought these were all cubic time algorithms, and beating cubic time is challenging in practice.
- Denoted "near-cubic time" operations.

Strassen's Famous Paper

- Strassen's 1969 paper had three algorithms, not one.
  - Matrix Multiplication
  - Matrix Inversion
  - Determinant
- The matrix multiplication runs in time $O(n^w)$ where $w = \log_2 7 \approx 2.807$.
- Strassen's Matrix Inversion Formula shows that matrix inversion *is no harder* than matrix multiplication.
- Reverse follows from:

$$\begin{bmatrix} I & A & 0 \\ 0 & I & B \\ 0 & 0 & I \end{bmatrix}^{-1} = \begin{bmatrix} I & -A & AB \\ 0 & I & -B \\ 0 & 0 & I \end{bmatrix}$$

- Note: Only 3 pages & changed Linear Algebra forever!

Strassen's Matrix Inversion Formula

$$A^{-1} = \begin{bmatrix} W & X \\ Y & Z \end{bmatrix}^{-1} = \begin{bmatrix} W^{-1} + W^{-1}XS^{-1}YW^{-1} & -W^{-1}XS^{-1} \\ -S^{-1}YW^{-1} & S^{-1} \end{bmatrix}$$

- Where $S = Z - YW^{-1}X$, is the Schur Complement of A with respect to W.
- The best procedure has 6 matrix multiplications, and 2 matrix inversions, or 8 near-cubic time operations.
- MAJOR PROBLEM: What if W or S is not invertible?!

The S Lemma

- If A and W are invertible, then the Schur Complement of A with respect to W is invertible also.
- A proof of this is found in *Introduction to Algorithms* by Cormen, Leiserson, Rivest and Stein. It only works for characteristic zero and is over a page.
- Instead, the following works over any field and was written by my advisor, Larry Washington.

$$\begin{bmatrix} I & 0 \\ -Y & I \end{bmatrix} \begin{bmatrix} W^{-1} & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} W & X \\ Y & Z \end{bmatrix} = \begin{bmatrix} I & W^{-1}X \\ 0 & \underbrace{Z - YW^{-1}X} \end{bmatrix}$$

- Thus taking the determinant

$$(\det I)(\det W)^{-1}(\det A) = (\det S)$$

- These are all non-zero on the left, so $\det S \neq 0$.

Las Vegas Algorithms

- This takes care of the invertibility of S, but what about W?
- Two options:
  - Failure.
    - Try the algorithm.
    - If at any step, W is singular, send the "abort" signal.
    - Never wrong, but only right with a particular probability.
  - Shuffling.
    - Try the algorithm.
    - If at any step, W is singular, shuffle A and repeat.
    - Expected number of shuffles can be calculated. No upper bound.
    - Never wrong, always right.
    - Expected running time can be calculated, worse-case is infinity.
- Distinct from Monte-Carlo, with fixed and known running time, but with errors.

Strassen's Matrix Inversion Formula

- What is the running time without shuffling?
- Suppose matrix multiplication is $cn^{2.807}$ operations.

$$
\begin{aligned}
I_G(n) &\sim 6M_{GG}(n/2) + 2I_G(n/2) \\
&\sim 6M_{GG}(n/2) + 12M_{GG}(n/4) + 4I_G(n/4) \\
&\sim 6M_{GG}(n/2) + 12M_{GG}(n/4) + 24M_{GG}(n/8) + 8I_G(n/8) \\
&\sim 6c\frac{n^w}{2^w} + 12c\frac{n^w}{4^w} + 24c\frac{n^w}{8^w} + 48c\frac{n^w}{16^w} + \cdots \\
&\sim 3cn^w \left[\Sigma_{i=1}^{i=\infty} \frac{2^i}{(2^i)^w}\right] \\
&\sim \frac{6}{5}cn^w
\end{aligned}
$$

- The LUP-Shuffle algorithm will do better.

Failure Analysis

- Suppose the algorithm is running on an $n \times n$ matrix, and will switch to classical methods at size $n_0 \times n_0$.
- The first layer has one iteration and thus has one chance of failure, the second layer has two iterations, thus two chances, etc...
- There are $1 + 2 + 4 + \ldots + 2^{i-1} = 2^i - 1$ chances of failure, where $i = \log_2(n/n_0)$.
- This means that there are $(n/n_0) - 1$ chances of failure. Total probability of abortion is

$$(1-p)^{\frac{n}{n_0}-1}$$

- ...where p is the probability that a random matrix is invertible. (Note p is almost independant of n and here we assume its completely independant. If n_0 is reasonable, this is not a problem).
- No matter what threshold you desire, this exceeds it for some n, and so another approach is needed.

How to Shuffle During Strassen's Matrix Inversion Formula

- See if W is invertible. If not, shuffle with P_x, and try again.
- If $WP_x = W'$ then $W^{-1} = P_x(W')^{-1}$.
- Multiplying by a permutation matrix is cheap (quadratic time).
- Let p be the probability that a random matrix is invertible.
- Instead of two inversions, we expect

$$\phi = 2 + \frac{1-p}{p}$$

With Shuffling

$$
\begin{aligned}
I_G(n) &\sim 6M_{GG}(n/2) + \phi I_G(n/2) \\
&\sim 6M_{GG}(n/2) + 6\phi M_{GG}(n/4) + \phi^2 I_G(n/4) \\
&\sim 6M_{GG}(n/2) + 6\phi M_{GG}(n/4) + 6\phi^2 M_{GG}(n/8) + \phi^3 I_G(n/8) \\
&\sim 6c\frac{n^w}{2^w} + 6\phi c\frac{n^w}{4^w} + 6\phi^2 c\frac{n^w}{8^w} + 6\phi^3 c\frac{n^w}{16^w} + \cdots \\
&\sim 6cn^w \left[\textstyle\sum_{i=1}^{i=\infty} \frac{\phi^{i-1}}{(2^i)^w}\right] \\
&\sim \frac{6}{7-\phi} cn^w \\
&\sim \frac{6}{7-2-(1-p)/p} cn^w \\
&\sim \frac{6}{6-1/p} cn^w
\end{aligned}
$$

What does that Imply?

- What does $\sim \frac{6}{6-1/p} cn^w$ mean?
- The series will only converge if $\phi/2^w = \phi/7$ is less than one.
- This happens if $p > 1/6$, but even for the smallest field, $GF(2)$, we have $p \approx 0.2888$, so this is not a problem.
- SANITY CHECK: If $p = 1$ (an infinite field), then we have $\sim (6/5)cn^w$ as before.
- If $p = 0.2888$ for $GF(2)$, we have 2.364, almost twice as bad.
- If $p = 0.9336$ for $GF(16)$, we have 1.217.
- If $p = 0.9961$ for $GF(256)$, we have 1.201, different only in the third decimal place!

Part Three: The LUP-Shuffle Algorithm

- An algorithm suitable for $GF(16)$, $GF(256)$, or other "medium-sized" and larger fields.
- Characteristic 2 is not required.
- Recursively break the matrix:

$$A = \begin{bmatrix} W & X \\ Y & Z \end{bmatrix}$$

- Based on a black-box matrix multiplication algorithm.
- If Strassen's Algorithm is used, $O(n^{2.807})$.
- Preserves Sparsity.

What is an LUP-factorization?

- Write $A = LUP$ where
  - L is lower-triangular.
  - U is upper-triangular.
  - P is a permutation matrix.
- Used very often in computer linear algebra implementations, along with its variant Cholesky Factorization, which will be mentioned later.
- Can be calculated along-the-way during Gaussian Elimination.
- Supplanted by the QR-Factorization in the last 20 years, but that algorithm does not work over finite fields, since not all non-zero vectors can have norm one. (A requirement for each column of Q).

What is the Benefit of Triangular Matrices?

- If $L\vec{x} = \vec{b}$, finding $\vec{x}$ is quadratic rather than cubic complexity.
- Thus solving $A\vec{x} = LUP\vec{x} = \vec{b}$ is of quadratic complexity, after LUP are found.
- If A is sparse, then L and U are sparse, while A^{-1} is in general, dense.
- Faster inversions and multiplies, as the next two slides will show.

Triangular-General Matrix Multiplication

$$\begin{bmatrix} W & X \\ 0 & Z \end{bmatrix} \begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} WA + XC & WB + XD \\ ZC & ZD \end{bmatrix}$$

- This requires six multiplications of smaller size, not eight if cubic, or seven if near-cubic (Strassen's Algorithm).
- Only XD and XC are general-general multiplications, the other four are general-triangular.
- It turns out, when recursively calculated as in the tabulations for Strassen's Matrix Inversion Formula that
  - Using Naive Matrix Multiplication, the triangular-general case is twice as fast as the general-general case.
  - Using Strassen's Matrix Multiplication, the triangular-general case is 3/2 as fast.
  - More details in the paper.

Triangular Matrix Inversion

$$\begin{bmatrix} W & X \\ 0 & Z \end{bmatrix}^{-1} = \begin{bmatrix} W^{-1} & -W^{-1}XZ^{-1} \\ 0 & Z^{-1} \end{bmatrix}$$

- Thus a triangular matrix inversion is an inversion of two smaller matrices, and two triangular-general multiplications.
- It turns out, when recursively calculated,
  - Using Naive Gaussian Elimination, the triangular-inversion case is six times as fast as the general-inversion case.
  - Using $O(n^{2.807})$ algorithms, the triangular-inversion case is $\sim (4/15)cn^w$ instead of $\sim (6/5)cn^w$, or 4.5 times faster.
  - More details in the paper.

The Concept

$$\begin{bmatrix} W & X \\ Y & Z \end{bmatrix} = \begin{bmatrix} L_1 & 0 \\ YP_1^{-1}U_1^{-1} & L_2 \end{bmatrix} \begin{bmatrix} U_1 & L_1^{-1}XP_2^{-1} \\ 0 & U_2 \end{bmatrix} \begin{bmatrix} P_1 & 0 \\ 0 & P_2 \end{bmatrix}$$

- . . . where $L_1U_1P_1 = W$,
- . . . and $L_2U_2P_2 = S = Z - YW^{-1}X = Z - YP_1^{-1}U_1^{-1}L_1^{-1}X$.
- (The Schur Complement again).

The Algorithm

- The algorithm proceeds as follows:
  1. Factor $W = L_1 U_1 P_1$.
  2. Invert P_1 (cheap).
  3. Invert L_1^{-1}.
  4. Invert U_1^{-1}.
  5. Multiply $Y P_1^{-1}$ (cheap).
  6. Multiply $(Y P_1^{-1}) U_1^{-1}$.
  7. Multiply $L_1^{-1} X$.
  8. Multiply $(Y P_1^{-1} U_1^{-1})(L_1^{-1} X)$.
  9. Subtract $S = Z - Y P_1^{-1} U_1^{-1} L_1^{-1} X$ (cheap).
  10. Factor $S = L_2 U_2 P_2$.
  11. Invert P_2 (cheap).
  12. Multiply $(L_1^{-1} X)(P_2^{-1})$. (cheap).
- Of the 12 operations, 5 are quadratic time or faster, and are marked cheap. The other 7 are near-cubic.

Summary

- To calculate the LUP-factorization of a matrix:
- Calculate two LUP-factorizations of half the size.
- Invert two *triangular* matrices.
- Multiply a general matrix with a *triangular* matrix (twice).
- Multiply a general matrix with a general matrix.
- This is seven near-cubic operations, not eight.
- Triangular inversion and Triangular-General matrix multiplication will be faster than normal.

No Shuffling

$$
\begin{aligned}
I_G(n) &\sim 2I_T(n/2) + 2M_{TG}(n/2) + M_{GG}(n/2) + 2L(n/2) \\
&\sim 2(4/15)M_{GG}(n/2) + 2(2/3)M_{GG}(n/2) + M_{GG}(n/2) + 2L(n/2) \\
&\sim (43/15)M_{GG}(n/2) + 2L(n/2) \\
&\sim (43/15)\left[M_{GG}(n/2) + 2M_{GG}(n/4)\right] + 4L(n/4) \\
&\sim (43/15)\left[M_{GG}(n/2) + 2M_{GG}(n/4) + 4M_{GG}(n/8)\right] + 8L(n/8) \\
&\sim (43/15)\left[c\frac{n^w}{2^w} + 2c\frac{n^w}{4^w} + 4c\frac{n^w}{8^w} + 8c\frac{n^w}{16^w} + \cdots\right] \\
&\sim (43/15)cn^w\left[\Sigma_{i=1}^{i=\infty}\frac{2^{i-1}}{(2^i)^w}\right] \\
&\sim \frac{43}{75}cn^w
\end{aligned}
$$

Chances of Failure

- Exactly as in the Strassen Matrix Inversion Formula, the algorithm fails if W or S is singular.
- The S Lemma guarantees S is non-singular if A and W are non-singular.
- Thus the probability of failure is identical to Strassen's Formula.
- And so shuffling must be used.
- If $AP_x = LUP$ then $A = LUPP_x^{-1}$.

With Shuffling

$$
\begin{aligned}
I_G(n) &\sim \phi I_T(n/2) + 2M_{TG}(n/2) + M_{GG}(n/2) + 2L(n/2) \\
&\sim \phi(4/15)M_{GG}(n/2) + 2(2/3)M_{GG}(n/2) + M_{GG}(n/2) + 2L(n/2) \\
&\sim (7/3 + 4\phi/15)M_{GG}(n/2) + 2L(n/2) \\
&\sim (7/3 + 4\phi/15)\left[M_{GG}(n/2) + 2M_{GG}(n/4)\right] + 4L(n/4) \\
&\sim (7/3 + 4\phi/15)\left[M_{GG}(n/2) + 2M_{GG}(n/4) + 4M_{GG}(n/8)\right] + 8L(n/8) \\
&\sim (7/3 + 4\phi/15)\left[c\frac{n^w}{2^w} + 2c\frac{n^w}{4^w} + 4c\frac{n^w}{8^w} + 8c\frac{n^w}{16^w} + \cdots\right] \\
&\sim (7/3 + 4\phi/15)cn^w\left[\Sigma_{i=1}^{i=\infty}\frac{2^{i-1}}{(2^i)^w}\right] \\
&\sim \frac{7/3 + 4\phi/15}{5}cn^w \\
&\sim \frac{35 + 4(2 + (1-p)/p)}{75}cn^w \\
&\sim (39/75 + 4/75p)cn^w
\end{aligned}
$$

With Shuffling

- What does $\sim (39/75 + 4/75p)cn^w$ tell us?
- If $p = 1$ for $\mathbb{Q}$ then we have $\sim (43/75)cn^w$, as without shuffling, or $\sim 0.5733cn^w$.
- If $p = 0.9961$ for $GF(256)$, then we have $\sim 0.5735cn^w$.
- If $p = 0.9336$ for $GF(16)$, then we have $\sim 0.5771cn^w$.
- If $p = 0.2888$ for $GF(2)$, then we have $\sim 0.7047cn^w$.

Speed Comparison

Field	Strassen's Alg	LUP-Shuffle Alg	Ratio
$\mathbb{Q}$	1.200	0.5733	2.093
$GF(256)$	1.201	0.5735	2.094
$GF(16)$	1.217	0.5771	2.109
$GF(2)$	2.364	0.7047	3.355

- The LUP-Shuffle algorithm is always at least twice as fast as Strassen's Matrix Inversion Formula.
- The $GF(2)$ coefficient divided by that for $\mathbb{Q}$ is 1.23 versus 1.97.
- This begs several questions
  - Why is LUP-Shuffle less sensitive to field size?
  - Why is $GF(16)$ behaving almost as if it is an infinite field?
  - Does this difference in coefficient matter?

Cross-Over Comparison

- Taking $GF(2)$ as an example, with $c = 7$ (naive implementation), then ...
- Strassen's Matrix Inversion Formula has $\sim 16.55n^w$.
- LUP-Shuffle has $\sim 4.933n^w$.
- Gaussian Elimination to Upper Triangular Form has $\sim n^3/2$.
- Solving $c_1 n^3 = c_2 n^w$ for n yields:
  - For Strassen's algorithm: $77,493,380 \times 77,493,380$ matrices.
  - For LUP-Shuffle: $144,704 \times 144,704$ matrices.
- So yes, the coefficient makes a difference.

Parallelization

- In Strassen's Matrix Inversion Formula, the eight near-cubic operations can be broken into six on one processor, two on another.
- In LUP-Shuffle, the seven near-cubic operations can be broken into five on one, and two on the other.
- The first processor is thus never idle, and idle 66% of the time for Strassen and 60% of the time for LUP-Shuffle.
- Thus the utilizations are 66% and 70% respectively.
- This gives LUP-Shuffle a 5.00% advantage.
- So in two-processor mode LUP-Shuffle would run 2.198 to 3.523 times faster rather than 2.093 to 3.355 times faster than Strassen for one processor.

Cholesky Factorization

- A Cholesky Factorization is an LU-factorization where $U = L^T$, or $A = LL^T$.
- This means some calculations can be saved, since U need not be found.
- Unlike LU-factorization, this algorithm must continue down to the 2×2 case.
- This means there are $n/2 - 1$ chances for failure, which is very high!
- There is no room for shuffling, except at the begining, since its not an $LL^T P$-factorization.
- Useful for infinite or very large fields where the probability is still very low, since virtually no random matrices are singular.

The Algorithm

$$\begin{bmatrix} W & X \\ X^T & Z \end{bmatrix} = \begin{bmatrix} L_1 & 0 \\ X^T L_1^{-T} & L_2 \end{bmatrix} \begin{bmatrix} L_1^T & L_1^{-1}X \\ 0 & L_2^T \end{bmatrix}$$

- ... where $W = L_1 L_1^T$ and ...

$$S = Z - X^T W^{-1} X = Z - (X^T L_1^{-T})(L_1^{-1} X) = L_2 L_2^T$$

- Requires 2 factorizations, 1 triangular inversion, 1 triangular-general multiplication, 1 $A^T A$ matrix multiplication.
- This is five near-cubic operations instead of seven!

One Caveat!

- Since $A = LL^T$ is the final factorization:
- This only can be done on symmetric matrices but...
- If A is not symmetric, note $B = A^T A$ will be and

$$B^{-1}A^T = A^{-1}A^{-T}A^T = A^{-1}$$

Part Four: The Method of Four Russians