Appears in SIGSAM Bulletin 22(2):41-49 (April 1988).

Analysis of the Binary Complexity of Asymptotically Fast Algorithms for Linear System Solving*

Brent Gregory and Erich Kaltofen

Department of Computer Science
Rensselaer Polytechnic Institute
Troy, New York 12180-3590
Inter-Net: kaltofen@cs.rpi.edu

Shortened Version

Introduction

Two significant developments can be distinguished in the theory of algebraic algorithm design. One is that of fast algorithms in terms of counting the arithmetic operations, such as the asymptotically fast matrix multiplication procedures and related linear algebra algorithms or the asymptotically fast polynomial multiplication procedures and related polynomial manipulation algorithms. The other is to observe the actual bit complexity when such algorithms are performed for concrete fields, in particular the rational numbers. It was discovered in the mid-1960s that for rational polynomial GCD operations the classical algorithm can lead to exponential coefficient size growth. The beautiful theory of subresultants by Collins [7] and Brown & Traub [5] explains, however, that the size growth is not inherent but a consequence of over-simplified rational arithmetic. Similar phenomena were observed for the Gaussian elimination procedure [2], [9].

In this article we investigate what size growth occurs in the asymptotically faster methods. We selected the matrix triangularization algorithm by Bunch & Hopcroft [6], [1], §6.4, as our paradigm. During Gaussian elimination of an integral matrix all occurring reduced rational numerators and denominators divide minors of the $n\times(n+1)$ augmented input matrix, so any intermediate number is at most roughly n times longer than the original entries. The asymptotically fast methods require asymptotically fast matrix multiplication. The fine structure of the best methods [8] is fairly complex and subject to change as improvements are made. We shall assume that the matrix multiplication algorithms are called with integral matrices, that is all rational entries are first brought to a common denominator. Since Strassen's [15] original method computes linear combinations of the entries, which has essentially the same effect, this assumption seems realistic. We then show that the Bunch & Hopcroft $n\times n$ matrix triangularization algorithm is worse than Gaussian elimination in that the intermediate numbers can be

* This material is based upon work supported by the National Science Foundation under Grant No. DCR-85-04391 (second author), and by an IBM Faculty Development Award (second author). This paper was presented at the poster session of the 1987 European Conference on Computer Algebra in Leipzig, East Germany.

roughly n^2 times longer than the original entries. In light of this negative result a modular version of the fast algorithm following McClellan [12] becomes computationally superior to the rational one. We strongly suspect that the fast Knuth-Schönhage polynomial GCD procedure [13], [3], behaves similarly. Again the modular version following Brown [4] would fix this problem.

We like to acknowledge that results similar to the ones discussed in this article were also obtained by Leighton [11].

Notation: By $\mathbf{K}$ we denote an arbitrary field, by $\mathbf{K}^{m\times n}$ the algebra of m by n matrices over $\mathbf{K}$. For a matrix A, $A(j_1, j_2, \cdots; k_1, k_2, \cdots)$ denotes the submatrix obtained from A by selecting the rows $j_1, j_2, \cdots$ and the columns $k_1, k_2, \cdots$. We also write ranges of rows/columns such as $j_1, \ldots, j_2$ meaning all j from inclusively j_1 to j_2. A full range is denoted by $*$. The elements of A are denoted by $a_{j,k}$. By I_n we denote the $n\times n$ identity matrix. Finally, by $M(n)$ we denote a function dominating the complexity of $n\times n$ matrix multiplication, currently at best $n^{\omega+o(1)}$, $\omega = 2.376\cdots$ [8].

Gaussian Elimination

We first state the result on the intermediate numbers during Gaussian elimination. In this and the following algorithms we assume that all principal minors are non-zero. Normally, this condition is enforced by permuting the columns during the algorithms. Since this has no influence on the size of the computed elements, we lose no generality by our assumption.

Algorithm *Gaussian Elimination*

Input: The algorithm triangularizes $U \in \mathbf{K}^{n\times n}$ in place. We superscribe the iteration counter to distinguish different versions of U. Initially, $U^{(0)} \leftarrow A \in \mathbf{K}^{n\times n}$, which is the matrix to be triangularized.

Output: $U^{(n-1)}$ in upper triangular form.

For $i \leftarrow 1, \ldots, n-1$ **Do**
  For $j \leftarrow i+1, \ldots, n$ **Do**
    For $k \leftarrow i+1, \ldots, n$ **Do**

$$u_{j,k}^{(i)} \leftarrow u_{j,k}^{(i-1)} - \frac{u_{j,i}^{(i-1)}\, u_{i,k}^{(i-1)}}{u_{i,i}^{(i-1)}}. \quad \square$$

Theorem: After the innermost loop completes, we have for all $1 \le i < n$, $i+1 \le j \le n$, $i+1 \le k \le n$,

$$u_{j,k}^{(i)} = \frac{\det(A(1,\ldots,i,j;\, 1,\ldots,i,k))}{\det(A(1,\ldots,i;\, 1,\ldots,i))}\,. \tag{1}$$

(See Gantmacher [10], Chapter II, for a proof.) $\square$

This leads not only to size bounds on the intermediate rationals by Hadamard's determinant inequality, but gives the following simple exact division rule: *The unreduced numerators and denominators of the new rational entries are divisible by their previous denominators.** Applying this rule, the algorithm then computes exactly the numerators and denominators in (1). A similar rule applies for the backward substitution process if one uses the triangularization to solve a linear system.

Fast Triangular Matrix Inversion

We now describe a division-free version of the asymptotically fast algorithm for computing the inverse of a triangular matrix [1], §6.3. This algorithm will be called by the asymptotically fast LU decomposition routine.

Algorithm *Triangular Inverse*

Input: $E \in D^{n\times n}$ upper triangular, $n = 2^r$, $r \geq 0$, D an integral domain.

Output: $E^* \in D^{n\times n}$ upper triangular such that $EE^* = (e_{1,1} \cdot \ \cdot \ \cdot e_{n,n})I_n$.

If $n = 1$ *Then Return* $E^* = I_1$.

$$\text{Let} \quad E_{1,1} := E(1,\ldots,\frac{n}{2}\,;\,1,\ldots,\frac{n}{2}), \quad E_{1,2} := E(1,\ldots,\frac{n}{2}\,;\,\frac{n}{2}+1,\ldots,n),$$

$$E_{2,2} := E(\frac{n}{2}+1,\ldots,n;\,\frac{n}{2}+1,\ldots,n).$$

Step R (Recursive calls): *Call the algorithm recursively with* $E_{1,1}$ *obtaining* $E^*_{1,1}$ *and* $E_{2,2}$ *obtaining* $E^*_{2,2}$.

Step M (Matrix multiplication): $\eta_1 \leftarrow e_{1,1} \cdot \ \cdot \ \cdot e_{n/2,n/2}$, $\eta_2 \leftarrow e_{n/2+1,n/2+1} \cdot \ \cdot \ \cdot e_{n,n}$.

$$\textit{Return} \quad E^* \leftarrow \begin{bmatrix} \eta_2 E^*_{1,1} & -E^*_{1,1}E_{1,2}E^*_{2,2} \\ 0 & \eta_1 E^*_{2,2} \end{bmatrix}. \quad \square$$

In terms of arithmetic operations in D, the running time $I(n)$ of this algorithm is estimated by

$$I(n) \leq 2I(\frac{n}{2}) + 2M(\frac{n}{2}) + O(n^2),$$

which gives $I(n) = O(M(n))$.

Fast LU Decomposition

* The discovery of this fact is difficult to trace in the literature. In recent times it has been treated by E. Bareiss, J. Edmonds, and D. K. Faddeev. Last century mathematicians sometimes associated with it include C. L. Dodgson, C. F. Gauss, C. Jordan, and J. J. Sylvester. Unfortunately, we have not found a convincing first source.

We now present a detailed description of the Bunch-Hopcroft algorithm. For later analysis purposes our algorithm changes the input matrix in place and is therefore more involved than the purely recursive version in [1], §6.4, which we recommend to the reader for an initial understanding of the algorithm.

Algorithm *Fast LU Decomposition*

Global Objects: L, $U \in \mathbf{K}^{n\times n}$, $\mathbf{K}$ a field, $n = 2^r$, $r \geq 0$. Furthermore, we will use an iteration counter i. L and U get updated in place, at which point i is incremented. We denote their versions as $L^{(i)}$, $U^{(i)}$. Initially, $L^{(0)} \leftarrow I_n$, $U^{(0)} \leftarrow A \in \mathbf{K}^{n\times n}$, which is the matrix to be decomposed. We assume that all principal minors of A are non-zero. The counter i merely helps to analyze the algorithm and does not participate in the computation.

Input: A row index j, $1 \leq j \leq n$, and a bandwidth $m = 2^s$, $0 \leq s < r$. Let the iteration counter on call be i_0. Figure 1 depicts a typical state at this point.

Output: Let the iteration counter on return be i_3. This call will have factored the sub-matrix

$$U^{(i_0)}(j,\ldots,\, j+m-1;\; j,\ldots,\, n) = L_3\; U_3,$$

where the lower triangular factor is embedded in $L^{(i_3)}$ as

$$L_3 := L^{(i_3)}(j,\ldots,\, j+m-1;\; j,\ldots,\, j+m-1) \in \mathbf{K}^{m\times m},$$

and the upper triangular factor in $U^{(i_3)}$ as

$$U_3 := U^{(i_3)}(j,\ldots,\, j+m-1;\; j,\ldots,\, n) \in \mathbf{K}^{m\times(n-j+1)}.$$

No other blocks in $L^{(i_0)}$ and $U^{(i_0)}$ will be changed by this call, so we always have $A = L^{(i)}\; U^{(i)}$.

If $m = 1$ *Then Return.* No update occurs. It is here that a column transposition of $U^{(i_0)}$ can make $u^{(i_0)}_{j,j} \neq 0$ in case a principal minor of A is 0.

Step U (Factor $U^{(i_0)}(j\,,\ldots,\; j+m/2-1;\; j\,,\ldots,\; n)$):
Call the algorithm recursively with j *and* $m/2$. Let i_1 be the value of i after this call. The updated blocks in L and U are

$$L_1 := L^{(i_1)}(j,\ldots,\, j+\frac{m}{2}-1;\; j,\ldots,\, j+\frac{m}{2}-1) \in \mathbf{K}^{\frac{m}{2}\times\frac{m}{2}}$$

and

$$U_1 := U^{(i_1)}(j,\ldots,\, j+\frac{m}{2}-1;\; j,\ldots,\, n) \in \mathbf{K}^{\frac{m}{2}\times(n-j+1)}.$$

Step Z (Zero-out $U^{(i_1)}(j+m/2\,,\ldots,\; j+m-1;\; j\,,\ldots,\; j+m/2-1)$):
Let

$$E := U_1(*;\, 1,\ldots,\, \frac{m}{2}) \in \mathbf{K}^{\frac{m}{2}\times\frac{m}{2}},$$

$$E' := U_1(*;\, \frac{m}{2}+1,\ldots,\, n-j+1) \in \mathbf{K}^{\frac{m}{2}\times(n-j+1-\frac{m}{2})},$$

and let

$$F := U^{(i_1)}(j+\frac{m}{2},\ldots,\, j+m-1;\; j,\ldots,\, j+\frac{m}{2}-1) \in \mathbf{K}^{\frac{m}{2}\times\frac{m}{2}},$$

$$F' := U^{(i_1)}(j+\frac{m}{2},\ldots,\, j+m-1;\; j+\frac{m}{2},\ldots,\, n) \in \mathbf{K}^{\frac{m}{2}\times(n-j+1-\frac{m}{2})}.$$

Notice that E is upper triangular, and that the blocks of $U^{(i_1)}$ corresponding to F and F' have not changed since version i_0. At this point

$$U^{(i_0)}(j,\ldots,\, j+m-1;\; j,\ldots,\, n) = \begin{bmatrix} L_1 & 0 \\ 0 & I_{m/2} \end{bmatrix} \times \begin{bmatrix} E & E' \\ F & F' \end{bmatrix}.$$

Call algorithm Triangular Inverse on E to obtain E^{-1}. *Compute* FE^{-1}. At this point L and U will be updated so we *increment* $i \leftarrow i+1$. The new value of i is $i_2 = i_1 + 1$. *Set*

$$L^{(i_2)}(j+\frac{m}{2},\ldots,\, j+m-1;\; j,\ldots,j+\frac{m}{2}-1) \leftarrow FE^{-1}, \tag{2}$$

$$U^{(i_2)}(j+\frac{m}{2},\ldots,\, j+m-1;\; j,\ldots,\, j+\frac{m}{2}-1) \leftarrow 0 \in \mathbf{K}^{\frac{m}{2}\times\frac{m}{2}},$$

$$U^{(i_2)}(j+\frac{m}{2},\ldots,\, j+m-1;\; j+\frac{m}{2},\ldots,\, n) \leftarrow F' - (FE^{-1})E'. \tag{3}$$

At this point we have

$$U^{(i_0)}(j,\ldots,\, j+m-1;\; j,\ldots,\, n) = \begin{bmatrix} L_1 & 0 \\ FE^{-1} & I_{m/2} \end{bmatrix} \times \begin{bmatrix} E & E' \\ 0 & F' - (FE^{-1})E' \end{bmatrix}.$$

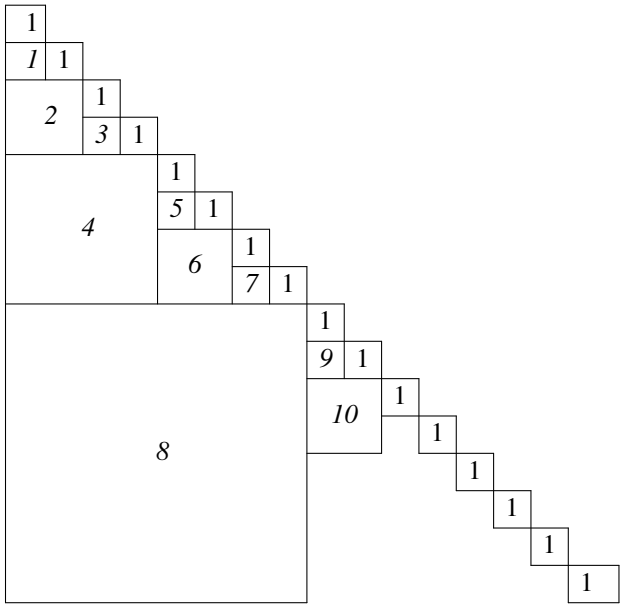
Step L (Factor $U^{(i_2)}(j+m/2\,,\ldots,\, j+m-1;\; j+m/2\,,\ldots,\, n)$):
Call the algorithm recursively with $j+m/2$ *and* $m/2$. □

The algorithm initially gets called with $j = 1$ and $m = n$. We then always have $i_0 = j-1$, $i_2 = j-1+m/2–1$, $i_3 = j-1+m-1$, as can be seen from Figure 1. In terms of arithmetic operations on an $n\times n$ matrix for input width m performed, we have

$$T_n(m) \le 2T_n(\frac{m}{2}) + I(\frac{m}{2}) + O(\frac{n}{m}\, M(\frac{m}{2})),$$

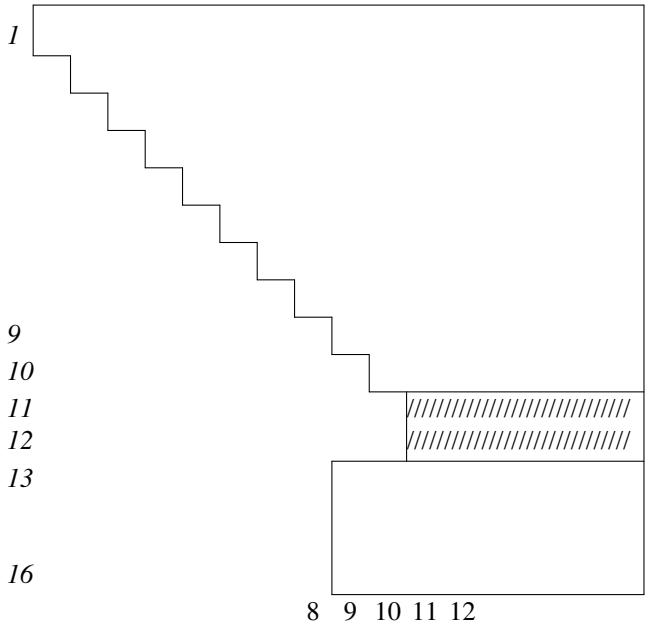
Figure 1: $L^{(10)}$ and $U^{(10)}$ upon call of algorithm Fast LU Decomposition with $n = 16$, $i = 10$, $j = 11$, $m = 2$. The sub-diagonal blocks in $L^{(10)}$ numbered in italics are filled in in the order given.

$L^{(10)}$



The submatrix of $U^{(10)}$ to be factored by this call is hashed.

$U^{(10)}$



The values of $\mu_j^{(10)}$ are determined by the incomplete fractal boundary of $U^{(10)}$. E. g. $\mu_9^{(10)} = 8$, $\mu_{10}^{(10)} = 9$, $\mu_{11}^{(10)} = 10$, $\mu_{12}^{(10)} = 10$, $\mu_{13}^{(10)} = 8$, $\mu_{14}^{(10)} = 8$.

which using a $O(m^\omega)$, $\omega > 2$, $m \times m$ matrix multiplication procedure gives $T_n(m) = O(nm^{\omega-1})$.

Analysis

As in the exact division algorithm by Gaussian elimination, we can give a formula for the entries in $U^{(i)}$. We define $\mu_j^{(i)}$ as the column index of that element in a row j of $U^{(i)}$ which is furthest to the right and which has been explicitly zeroed by step Z. If no such element exists, $\mu_j^{(i)} = 0$. Observe that $\mu_j^{(i)} = j - 1$ for $i \geq j - 1$. Also refer to Figure 1 for an example.

Theorem: After step Z in the Fast LU decomposition algorithm,

$$u_{j,k}^{(i)} = \frac{\det(A(1,\ldots,\mu_j^{(i)}, j;\ 1,\ldots,\mu_j^{(i)}, k))}{\det(A(1,\ldots,\mu_j^{(i)};\ 1,\ldots,\mu_j^{(i)}))} \tag{4}$$

for all $1 \leq i \leq n-1$, $1 \leq j \leq n$, $\mu_j^{(i)} < k \leq n$. □

Although (4) is the expected generalization of (1), the intermediate numerators and denominators become much larger now. The reason is that the matrix multiplication for (2) and (3) requires us, by the assumption made in the introduction, to bring the rational entries of E, E', and F on a common denominator. In the worst case, for $j = n/2 + 1$, $m = n/2$, we have

$$E' = U^{(n/2-1)}(1,\ldots,\frac{n}{2};\ \frac{n}{2}+1,\ldots,n).$$

The common denominator of its entries is

$$\prod_{j=1}^{n/2-1} \det(A(1,\ldots,j;\ 1,\ldots,j)), \tag{5}$$

which if the original matrix A has integer entries of length l can be an integer of length $O(n^2 l)$. Similar growth already occurs in the inversion E^{-1}, but the entries of F have a 'small' denominator. Over an abstract integral domain (5) is the best common denominator, but for specific domains such as the integers one could, of course, choose the GCD of the factors in (5). However, for random matrices this does not help much. The reason is that as polynomials in $\mathbf{Z}[a_{1,1}, \ldots, a_{n,n}]$ the factors are relatively prime polynomials, because determinants are irreducible polynomials [16], §23, Exercise 3. Now a random integral evaluation is likely to preserve relative primeness. For an explanation of this phenomenon we refer to [14].

As we have shown, exact rational arithmetic during the fast algorithms for linear system solving can lead to unexpected size growth. A remedy for this problem can be a modular approach, that is solving the system modulo prime numbers and then computing the rational solution by Chinese remaindering [12]. One gets the following theorem.

Theorem: Let $Ax = b$ be an $n \times n$ non-singular linear system in which A and b have integral entries of lenght at most l. Then one can find the rational solution vector x in $(nl)^{1+o(1)} n^{\omega+o(1)}$ binary steps. □

This theorem can also be generalized to singular systems, both in the over- and underdetermined case. Notice that the Fast LU Decompostion algorithm allows to compute the rank of an $m \times n$ matrix, $m \leq n$, in $O(m^{\omega-1} n)$ arithmetic operations.

Conclusion

The main result in this paper is negative, in that we show that one cannot unify exact division methods in computational algebra such as the Exact Division Gaussian elimination or the Subresultant PRS algorithm with asymptotically fast methods such as the Bunch-Hopcroft LU decomposition algorithm or the Knuth-Schönhage polynomial GCD algorithm. In gaining an asymptotic speed-up in terms of arithmetic operation count one significantly increases the size of intermediately computed fractions. For specific domains, such as the rational numbers, this growth can be controlled by modular techniques.

Note added on September 8, 2006: In the Theorem on page 2, the ranges of j and k were corrected.

References

1. Aho, A., Hopcroft, J., and Ullman, J., *The Design and Analysis of Algorithms,* Addison and Wesley, Reading, MA (1974).

2. Bareiss, E. H., "Sylvester's identity and multistep integers preserving Gaussian elimination," *Math. Comp.* **22**, pp. 565-578 (1968).

3. Brent, R. P., Gustavson, F. G., and Yun, D. Y. Y., "Fast solution of Toeplitz systems of equations and computation of Padé approximants," *J. Algorithms* **1**, pp. 259-295 (1980).

4. Brown, W. S., "On Euclid's algorithm and the computation of polynomial greatest common divisors," *J. ACM* **18**, pp. 478-504 (1971).

5. Brown, W. S. and Traub, J. F., "On Euclid's algorithm and the theory of subresultants," *J. ACM* **18**, pp. 505-514 (1971).

6. Bunch, J. R. and Hopcroft, J. E., "Triangular factorization and inversion by fast matrix multiplication," *Math. Comp.* **28**, pp. 231-236 (1974).

7. Collins, G. E., "Subresultants and reduced polynomial remainder sequences," *J. ACM* **14**, pp. 128-142 (1967).

8. Coppersmith, D. and Winograd, S., "Matrix multiplication via arithmetic progressions," *Proc. 19th Annual ACM Symp. Theory Comp.,* pp. 1-6 (1987).

9. Edmonds, J., "Systems of distinct representatives and linear algebra," *J. Research National Bureau Standards B* **71B**, pp. 241-245 (1967).

10. Gantmacher, F. R., *The Theory of Matrices, Vol. 1,* Chelsea Publ. Co., New York, N. Y. (1960).

11. Leighton, F. T., "Gaussian elimination over the rationals," Unpublished Manuscript, Applied Math. Dept., MIT (1980).

12. McClellan, M. T., "The exact solution of systems of linear equations with polynomial coefficients," *J. ACM* **20**, pp. 563-588 (1973).

13. Moenck, R. T., "Fast computation of GCDs," *Proc. 5th ACM Symp. Theory Comp.,* pp. 142-151 (1973).

14. Schönhage, A., "Probabilistic computation of integer GCDs," *J. Algorithms* (to appear).

15. Strassen, V., "Gaussian elimination is not optimal," *Numerische Mathematik* **13**, pp. 354-356 (1969).

16. Waerden, B. L. van der, *Modern Algebra,* F. Ungar Publ. Co., New York (1953).