On parallel block algorithms for exact triangularizations

Jean-Guillaume Dumas [a] Jean-Louis Roch [b]

[a] *Laboratoire de Modélisation et Calcul*
B. P. 53 – 51, av. des Mathématiques,
38041 Grenoble, France.

[b] *Laboratoire Informatique et Distribution*
projet APACHE, CNRS–INRIA–INPG–UJF.
ZIRST - 51, av. Jean Kuntzmann
38330 Montbonnot Saint-Martin, France.

Abstract

We present a new parallel algorithm to compute an exact triangularization of large square or rectangular and dense or sparse matrices in any field. Using fast matrix multiplication, our algorithm has the best known sequential arithmetic complexity. Furthermore, on distributed architectures, it drastically reduces the total volume of communication compared to previously known algorithms. The resulting matrix can be used to compute the rank or to solve a linear system. Over finite fields, for instance, our method has proven useful in the computation of large Gröbner bases arising in robotic problems or wavelet image compression.

Key words: Parallel Block triangularization, Exact LU factorization of rectangular matrices, Symbolic rank computation, Sparse matrices, Galois Finite fields, BLAS, Fast symbolic matrix multiplication.

1 Introduction

In this article, we study the parallelization of the exact LU factorization of matrices with arbitrary field elements. The matrix can be singular or even rectangular with dimension $m \times n$. Our main purpose is to compute the rank of large matrices. Therefore, we relax the conditions on L in order to obtain a

Email addresses: `Jean-Guillaume.Dumas@imag.fr` (Jean-Guillaume Dumas), `Jean-Louis.Roch@imag.fr` (Jean-Louis Roch).

Preprint submitted to Elsevier Science *20 June 2002*

TU factorization, where U is $m \times n$, upper triangular as usual and T is $m \times m$, block sparse (with some "T" patterns).

Exact triangularization arises in various applications and especially in computer algebra. For instance, one of the main tools for solving algebraic systems is the computation of Gröbner bases[1]. Actual methods to compute such standard bases use modular triangularization of large sparse rectangular matrices [2]. Among other applications of symbolic LU are combinatorics [3], fast determinant computation, Diophantine analysis, group theory and algebraic topology via the computation of the integer Smith normal form (an integer diagonal canonical form [4]).

A first idea is to use a parallel direct method [5, chapter 11] on matrices stored by rows (respectively columns). There, at stage k of the classical Gaussian elimination algorithm, eliminations are executed in parallel on the $n-k-1$ remaining rows. This method does not directly enable efficient modular computations which are of low cost. Then, gathering the operations to obtain a larger grain size is necessary.

The next idea is then to mimic numerical methods and use sub-matrices. This way, as the computations are local, it is possible to take advantage of the cache effects as in the BLAS numerical libraries [6]. Now, the problem is that usually, for symbolic computation, these blocks are singular. To solve this problem one has mainly two alternatives. One is to perform a dynamic cutting of the matrix and to adjust it so that the blocks are reduced and become invertible. Such a method is shown by Ibarra et al. in [7; 8], and studied in detail in [9, Chapter 2]. Their algorithm (LSP) groups *rows* into two regions [10, Problem 2.7c]. A recursive process is then used to compute the rank of the first region. Then, depending on this rank, the cutting is modified and the algorithm pursues with a new region. This way, Ibarra et al. were able to build the first algorithm computing the rank of an $m \times n$ matrix with arithmetic complexity $\mathcal{O}(m^{\omega-1}n)$, where the complexity of matrix multiplication is $O(m^{\omega})$.

Unfortunately, their method is not so efficient in parallel: it induces synchronizations and significant communications at each stage in order to compute the block redistribution.

We therefore propose another method, called *TURBO*, using *static blocks* (which might be *singular*) in order to avoid these synchronizations and redistributions. Our algorithm also has an optimal sequential arithmetic complexity and is able to avoid as much as a third of the communications.

A preliminary version of this paper appeared in [11]. Here we include a complete asymptotic analysis, give sharper bounds on the number of needed arithmetic operations and offer more experimental results.

The paper is organized as follows. In Section 2, we detail this new recursive block algorithm. This presentation is followed, in Sections 3 and 4, by asymptotic arithmetic and communication cost analyses. Finally, in Section 5, practical performance is shown on matrices involved in the computation of Gröbner bases and Homology groups of simplicial complexes [12; 13].

2 A new block algorithm

2.1 TURBO algorithm

In *TURBO*, the elementary operation is a block operation and not a scalar one. In addition, the cutting of the matrix in blocks is carried out before the execution of the algorithm and is not modified, in order to limit the volume of communications. We choose to describe the algorithm in a recursive way to simplify its theoretical study. The threshold of recursive cutting is then settled as the initial structure of the matrix is decided. Now take a $2m \times 2n$ matrix A over a field $\mathbb{F}$. Our method recursively divides the matrix into four regions:

$$A = \begin{bmatrix} NW & NE \\ SW & SE \end{bmatrix}$$

Local TU factorizations are then performed on each block. The method is applied recursively until the region size reaches a given threshold. We show here the algorithm for only one iteration. The factorization is done *in place*, i.e. the matrix A in input is not copied: the algorithm modifies its elements progressively.

The following code computes the upper triangular form U. It can easily be completed in order to also compute the matrix T such that $A = TU$. To illustrate the algorithm we show figures representing the matrix after each step. In these figures we supposed that the initial cutting of the matrix was in $10 \times 10 = 100$ blocks.

Algorithm *TURBO*: TU Recursive factorization with Blocks.

Input : – a matrix $A \in \mathbb{F}^{2m \times 2n}$, of rank r.
Output: – A, modified in place as an upper triangular matrix with invertible leading principal minor $r \times r$.

Step 1. Recursive TU triangularization in NW

Compute $L_1 \in \mathbb{F}^{m \times m}$ lower triangular, $U_1 \in \mathbb{F}^{q \times q}$ upper triangular with

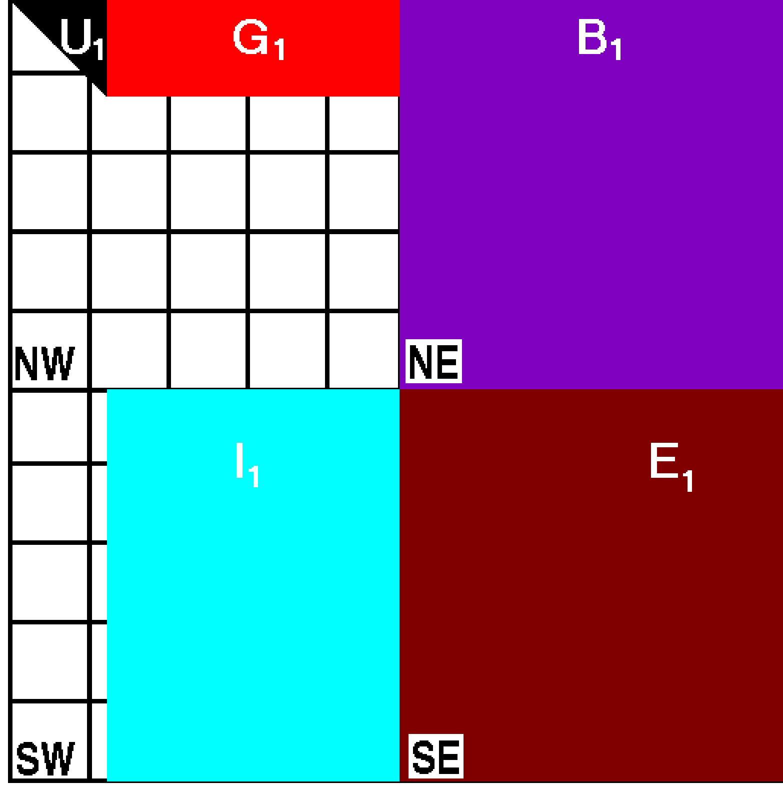

Fig. 1. Matrix A after step 1

rank q and G_1 such that

$$L_1 \times NW = \begin{bmatrix} U_1 & G_1 \\ 0 & 0 \end{bmatrix}$$

The NE region can now be updated: $B_1 = L_1 \times NE$. Simultaneously, zeroes under U_1 in SW are computed.
Let $SW_{(,1..q)}$ stands for the first q columns of SW; then, $N_1 = -SW_{(,1..q)} \times U_1^{-1} \in \mathbb{F}^{m \times q}$ is computed.
Finish by the update of the SW region: $I_1 = SW_{(,q+1..n)} + N_1 \times G_1$ and the first rows of NE are multiplied by N_1 in order to update SE: $E_1 = SE + N_1 \times B_{1(1..q,)}$.

Step 2. Recursive TU triangularization in SE
Compute $L_2 \in \mathbb{F}^{m \times m}$ lower triangular, $V_2 \in \mathbb{F}^{p \times p}$ upper triangular with rank p and E_2 such that

$$L_2 \times E_1 = \begin{bmatrix} V_2 & E_2 \\ 0 & 0 \end{bmatrix}$$

Then, as in step 1, SW is updated using

$$\begin{bmatrix} I_2 \\ F_2 \end{bmatrix} = L_2 \times I_1$$

and zeroes over V_2 are computed:
i.e. $N_2 = -B_{1(q+1..m,1..p)} \times V_2^{-1}$
Then, $H_2 = B_{1(q+1..m,p+1..n)} + N_2 \times E_2$ and
$O_2 = NW_{(q+1..m,q+1..n)} = N_2 \times I_2$.

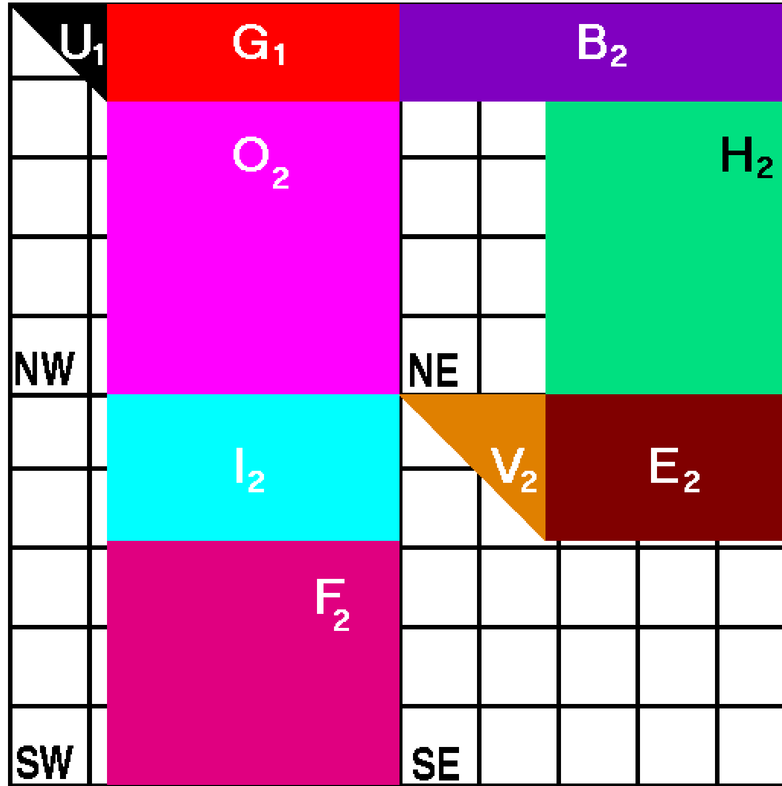

Fig. 2. Matrix A after step 2

Step 3. Parallel recursive TU in SW and NE
Compute L_3, $D_3 \in \mathbb{F}^{d \times d}$ with rank d, F_3 and M_3, $C_3 \in \mathbb{F}^{c \times c}$ with rank c, H_3 such that

$$L_3 \times F_2 = \begin{bmatrix} D_3 & F_3 \\ 0 & 0 \end{bmatrix} \text{ and } M_3 \times H_2 = \begin{bmatrix} C_3 & H_3 \\ 0 & 0 \end{bmatrix}$$

Then NW is updated

$$\begin{bmatrix} O_3 \\ X_2 \; T_2 \end{bmatrix} = M_3 \times O_2$$

and zeroes over D_3 are computed using $N_3 = -X_2 \times D_3^{-1}$.

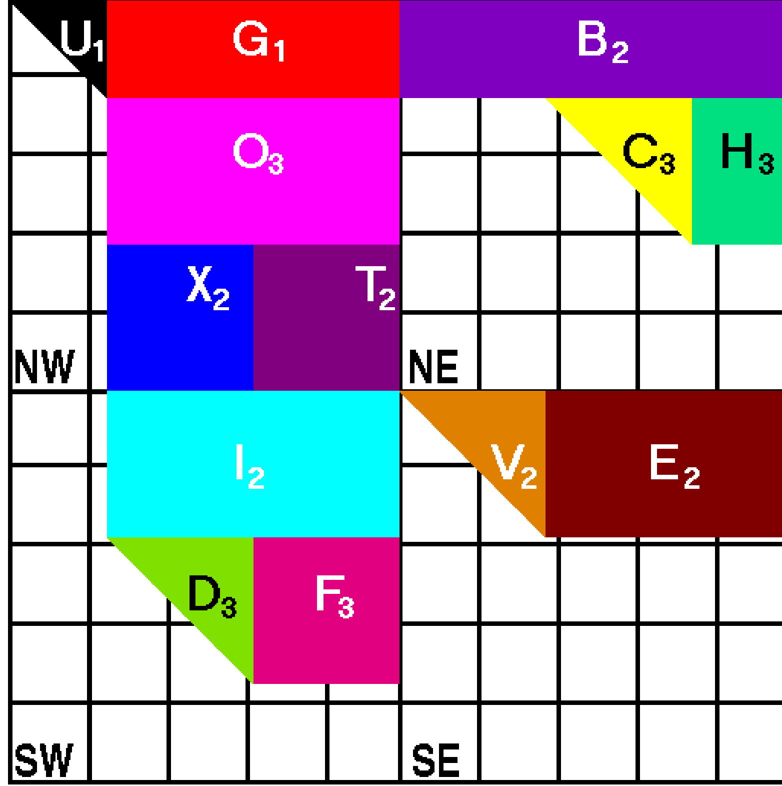

Fig. 3. Matrix A after step 3

This step ends with $T_3 = T_2 + N_3 F_3$.
Of course, it is possible to create zeroes to the left of C_3 instead of over D_3 in NW. On the one hand, the choice can be made in view of the respective dimensions of the blocks. The smallest block, i.e. the one inducing the maximum number of zeroes and consequently the minimum volume of communication, is chosen. On the other hand, a parallel version could choose the first ready block instead.

Step 4. Small recursive TU in NW again
Compute L_4, $Z_4 \in \mathbb{F}^{z \times z}$ with rank z and T_4, such that

$$L_4 \times T_3 = \begin{bmatrix} Z_4 & T_4 \\ 0 & 0 \end{bmatrix}$$

Step 5. Virtual row permutations
I_2–V_2–E_2 can be inserted between U_1–G_1–B_2 and O_3–C_3–H_3.
Z_4–T_4 can be moved under F_3.

Step 6. Virtual column permutations

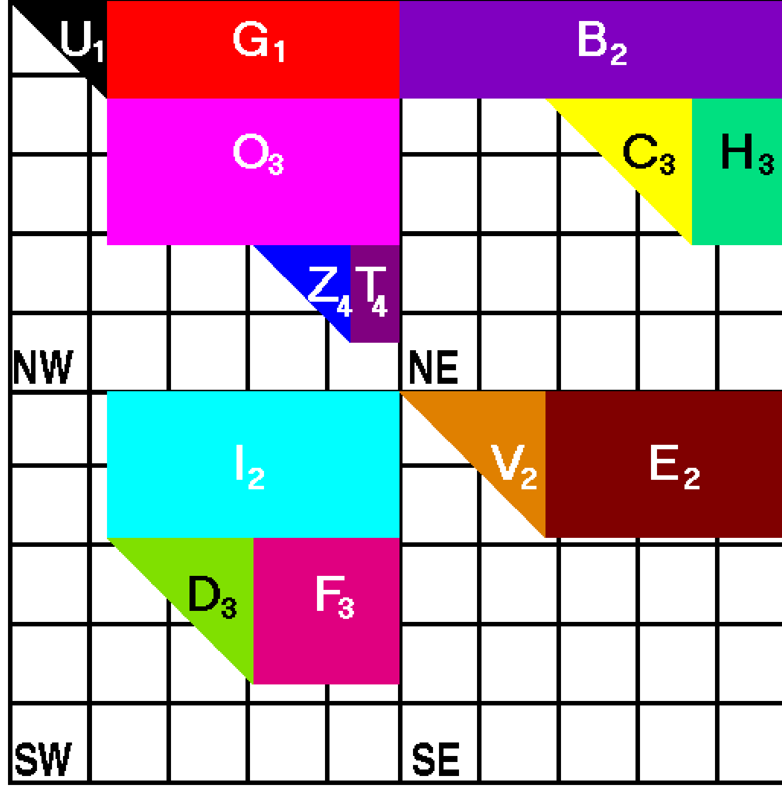

Fig. 4. Matrix A after step 4

B_2–V_2–E_2–C_3–H_3 can be inserted between U_1 and G_1–I_2–O_3–D_3–F_3–Z_4–T_4.

Step 7. Rank
r = q + p + c + d + z;

Numbers, in Figures 1, 2, 3, 4 and 5 match the last modification step. The whole algorithm is a variant of this one preserving the intermediate matrices L_i. Moreover, as steps 5 and 6 are only virtual, a permutation vector is computed in order to enable next recursive phases; Figure 5, shows the matrix if those two steps are performed.

Now, Figure 6 shows the data dependency graph between the different computations. This graph can be recursive for the whole algorithm, as for the large multiplication tasks. Besides, for those, fast parallel matrix multiplication is applied.

The major interest of this algorithm, apart from enabling fast matrix arithmetic, is the communication savings. Indeed, most of the operations are local

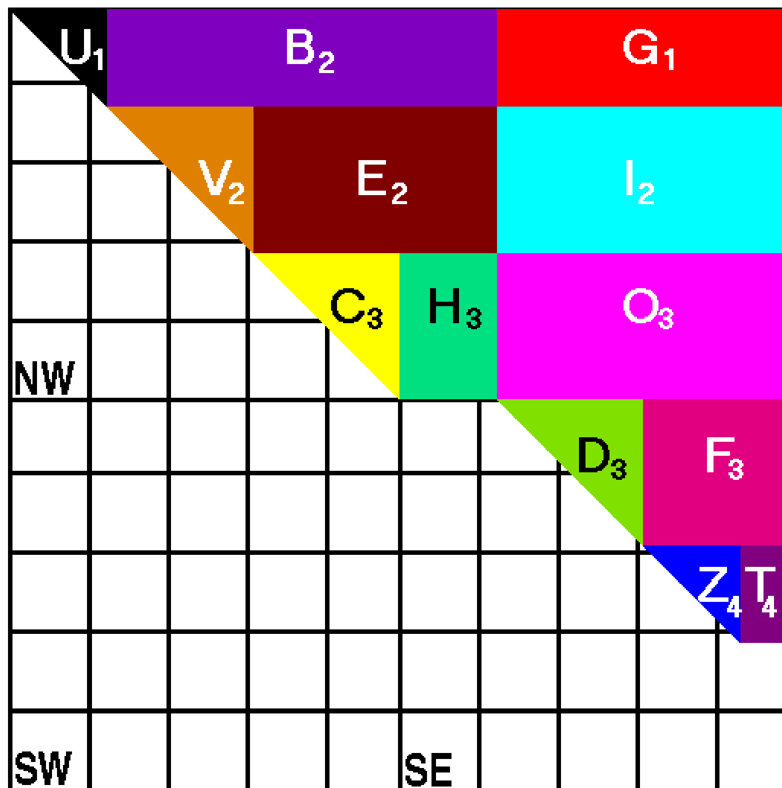

Fig. 5. Matrix A after virtual step 6

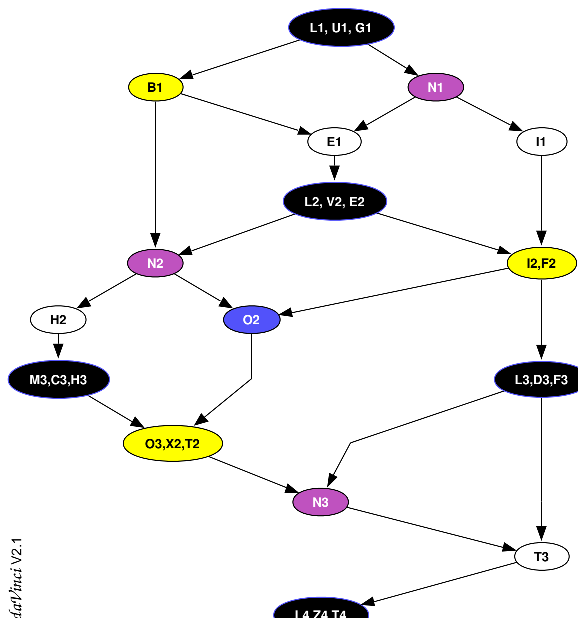

Fig. 6. Data dependency graph for the parallel block recursive algorithm

and only the updating matrices are sent to the other regions. Further, as the computations are local and not redistributed, it is possible to efficiently take advantage of the cache effects inside the blocks (in a BLAS way, see [14], where an efficient implementation of finite field linear algebra subroutines on top of numerical BLAS is proposed).

2.2 Effective weight of the steps

In practice, Table 1 shows the successive ranks for different matrices arising in Gröbner bases computations (robot24_m5, rkat7_m5, f855_m9 and cyclic8_m11 [15]) and, using integer matrix Smith normal form, arising when computing Homology groups of simplicial complexes (mk***i***.b***k*** are matching complexes and ch***i-j***.b***k*** are chess-board complexes [16; 17]).

Matrix	$2m \times 2n$	q	p	c	d	z
ch5-5.b2	600x200	100	55	21	0	0
mk9.b2	1260x378	189	102	52	0	0
ch6-6.b2	2400x450	225	105	85	0	0
ch4-4.b2	96x72	24	23	7	3	0
ch5-5.b3	600x600	156	158	70	40	0
mk9.b3	945x1260	286	334	136	119	0
robot24_m5	404x302	54	141	10	57	0
rkat7_m5	694x738	95	239	130	108	39
f855_m9	2456x2511	123	1064	189	164	791
cyclic8_m11	4562x5761	392	1927	521	354	709

Table 1
Successive ranks in the block recursive algorithm

We see that the last block is often all zero ($z = 0$), and that the other four ranks are quite homogeneously balanced on the average.

The next sections study the arithmetic complexities and memory costs of our technique.

3 Arithmetic complexity

In this section, we show that our algorithm has an arithmetic complexity similar to the best known ones. We show also that its parallel arithmetic complexity is analogous to those of the most efficient parallel algorithms.

3.1 Sequential cost

ω is the exponent of the complexity of matrix multiplication (i.e. 3 for the classical multiplication or $\log_2(7) \approx 2.807355$ for Strassen's [18], the actual record being below 2.375477 [19] and the lower bound being 2). We will therefore denote the cost of the multiplication of two matrices, of respective dimensions $2m \times 2n$ and $2n \times 2l$, by $M(h) = \mathcal{O}(h^\omega)$ for $h = max\{2m; 2n; 2l\}$.

THEOREM 3.1. *Let $T_1(h)$ be the sequential arithmetic complexity of algorithm* TURBO *for a rectangular matrix of higher dimension h. Then,*

$$T_1(h) \le \frac{29}{4(2^{\omega-1}-1)} M(h) + 2h^2 = \mathcal{O}(h^\omega).$$

Proof. To prove this theorem, the 5 triangularizations have to be gathered into two groups such that each one of these two groups has a total rank less than $\frac{h}{2}$. An induction process can then be applied to the respective costs in order to achieve the given upper bound:
Following the dependency Graph 6 and the algorithm, we see that *TURBO* requires 4 multiplications, 1 triangular inversion and 2 additions for step 1. On step 2, 4 multiplications, 1 triangular inversion and only 1 addition are required. We end by step 3 where 2 multiplications, 1 triangular inversion and 1 addition are needed. This sums up to 10 multiplications, 3 inversions and 4 additions of matrices of size smaller than $\frac{h}{2}$. Moreover, if the multiplication cost is $M(h) \le Kh^\omega$, triangular inversion is bounded by $\frac{3}{2}M(h)$ [20, theorem 6.2]. Then the cost of our algorithm is bounded as follows (q, p, c, d and z are the respective ranks of U_1, V_2, C_3, D_3 and Z_4):

$$\begin{aligned} T_1(h) \le\ & T_1(p) + T_1(q) + T_1(c) + T_1(d) \\ & + T_1(z) + \frac{29}{2} K \frac{h^\omega}{2^\omega} + h^2 \end{aligned} \tag{1}$$

Now suppose, that $T_1(x) \le K'x^\omega + 2x^2, \forall x < h$, for some K'. Then, by induction we have:

$$\begin{aligned} T_1(h) \le\ & K'(q^\omega + c^\omega + z^\omega + p^\omega + d^\omega + 2\frac{h^\omega}{2^\omega}) \\ & + 2(q^2 + c^2 + z^2 + p^2 + d^2) + h^2. \end{aligned} \tag{2}$$

Also, as $\omega \ge 2$ and all the intermediate ranks are non negative, we bound $q^\omega + c^\omega + z^\omega$ by $(q + c + z)^\omega$ and $p^\omega + d^\omega$ by $(p + d)^\omega$. Besides, using the ranks

locality, we have:

$$q + c + z \leq \frac{h}{2} \qquad \text{and} \qquad p + d \leq \frac{h}{2} \tag{3}$$

$$q + d + z \leq \frac{h}{2} \qquad \text{and} \qquad p + c \leq \frac{h}{2} \tag{4}$$

Thus, with Relations 3, we get

$$T_1(h) \leq (\frac{4K' + 29K}{2^{\omega+1}})h^\omega + 2h^2$$

Therefore we can take $K' = \frac{29}{2^{\omega+1}-4}K$ and, as $T_1(2) = 3$, the induction and the theorem are proven. □

The theorem shows that when using classical multiplication, our algorithm requires then the equivalent of only $\frac{29}{12} \approx 2.416667$ matrix multiplications. On the other hand, the arithmetic complexity of our algorithm is the best known complexity for this problem in terms of "big-Oh" [8, Theorem 2.1]. But Ibarra's algorithm (LSP) has a different bound: $\frac{3}{2^{\omega-1}-2}$. We compare these two bounds for different kinds of matrix multiplications in Table 2.

ω	TURBO	LSP
3	2.416667	1.5
2.807375	2.899943	1.999935
2.375477	4.546775	5.045945

Table 2
Number of arithmetic operations for TURBO and LSP

The table shows that for most of the practical algorithms, our method requires a few more operations. But, unlike our method, Ibarra's algorithm groups *rows* into two regions. Then, depending on the rank of the first region, the matrix structure is *modified*. Using our block cutting, we instead guarantee that all the accesses are local, thus enabling faster computations.

3.2 Parallel cost

In the parallel case, the situation is different. We obtain only a linear complexity. Considering that parallel triangular matrix multiplication and inversion costs are logarithmic (lower than $K_\infty \log_2{}^2(h)$) we have the following result:

THEOREM 3.2. *Let $T_\infty(h)$ be the parallel arithmetic complexity of algorithm* TURBO *for a rectangular matrix of higher dimension $h \geq 22$. Then,*

$$T_\infty(h) \leq 3K_\infty h = \mathcal{O}(h).$$

As in the sequential case, the idea of this theorem is to gather triangularizations in order to have groups of total rank less than $\frac{h}{2}$.

Proof. Here, $T_\infty(c)$ and $T_\infty(d)$ are parallel. Without loss of generality, we suppose that $c \geq d$, and then the difference with the sequential inequality is that $T_\infty(d)$ is shadowed by $T_\infty(c)$. Then, let $h_1 = q$, $h_2 = p$, $h_3 = c$ and $h_4 = z$. As the parallel multiplication cost is bounded by $K_\infty \log_2{}^2(h)$, we have:

$$\begin{aligned} T_\infty(h) &= K_\infty \log_2{}^2(h) + T_\infty(q) + T_\infty(p) + T_\infty(c) + T_\infty(z) \\ &= K_\infty \log_2{}^2(h) + \sum_{i=0}^{4} T_\infty(h_i) \end{aligned}$$

Therefore, developing *one more* recursive phase, we get, assuming for now that the four ranks are non zero:

$$\begin{aligned} T_\infty(h) = K_\infty\Big(&\log_2{}^2(h) + \log_2{}^2(q) + \log_2{}^2(c) \\ &+ \log_2{}^2(p) + \log_2{}^2(z)\Big) + \sum_{i=0}^{4}\sum_{j=0}^{4} T_\infty(h_{i;j}) \end{aligned} \tag{5}$$

where $T_\infty(h_{2;3})$, for instance, is the cost connected the rank of block C of the second recursive phase, inside block V of the first recursive phase.

Now, as in the sequential theorem, costs are gathered in order to have groups where the rank sum is less than $\frac{h}{2}$. Thus, we have $q + c \leq \frac{h}{2}$. First, we suppose that $q \geq 1$ and $c \geq 1$. Then $\log_2{}^2(q) + \log_2{}^2(c)$ is maximal when $q = c = \frac{h}{4}$, as soon as $h \geq 16$. Else, if one of these two ranks is zero, then the cost of the group is bounded by $\log_2{}^2(\frac{h}{2})$. But $\log_2{}^2(\frac{h}{2}) \leq 2\log_2{}^2(\frac{h}{4})$ as soon as $h \geq 2^{3+\sqrt{2}} \approx 21.3212$. Therefore, asymptotically, $(\log_2{}^2(q) + \log_2{}^2(c)) \leq 2\log_2{}^2(\frac{h}{4})$ In the same manner we have $(\log_2{}^2(p) + \log_2{}^2(z)) \leq 2\log_2{}^2(\frac{h}{4})$. Therefore, Formula 5 becomes

$$T_\infty(h) \leq K_\infty\Big(\log_2{}^2(h) + 4\log_2{}^2(\frac{h}{4})\Big) + \sum_{i,j=0}^{4} T_\infty(h_{i,j}) \tag{6}$$

We finish the proof by gathering and bounding again recursively in the same manner:

$$T_\infty(h) \leq K_\infty \sum_{i=0}^{\log_4(h)} 4^i \log_2{}^2(\frac{h}{4^i}) \leq K_\infty \frac{80}{27} h \leq 3K_\infty h.$$

□

Therefore, the theoretical complexity is linear, while rank computation is in NC^2: using $\mathcal{O}(n^{4.5})$ processors, the computation can be performed with parallel time $\mathcal{O}(\log_2{}^2(n))$ [21]. However, the best known parallel algorithm with

optimal sequential time also achieves a parallel linear time [8; 10]. But, in practice, our technique is more interesting as it preserves locality. Further, we will see that it reduces the volume of communications on distributed architectures.

4 Scheduling, blocking and communications

We first consider the execution of algorithm *TURBO* on a PRAM [22] with P processors. For a given matrix, once the computational dependency Graph 6 is known, Brent's principle [22; 10], ensures that execution can be performed in time $T_P(h) \leq T_\infty(h) + \frac{T_1(h)}{P}$. However, here, the graph is dynamically generated since the rank of the related sub-matrices is only known at execution time. Thus, Brent's bound cannot be applied for classical rank algorithms: the overhead σ [23] introduced for the scheduling (allocation of tasks to idle processors [24]) is to be considered and leads to:

$$T_P(h) \leq T_\infty(h) + \frac{T_1(h)}{P} + \sigma(h) \tag{7}$$

In order to achieve asymptotically fast execution on P processors, σ must be reduced. A blocking technique is classically used to this end. When a block of size smaller than a given value k is encountered, its rank is computed using the optimal sequential algorithm. Then the number of tasks is clearly bounded by $T_1(\frac{h}{k})$. Hence, the scheduling overhead is

$$\sigma\left(\frac{h}{k}\right) = O\left(\left(\frac{h}{k}\right)^\omega\right).$$

Choosing k large enough with respect to $\frac{T_1}{h^\omega}$ ensures asymptotically optimal execution in time

$$T_P(h) \leq T_1(k)T_\infty(h) + \frac{T_1(h)}{P} + \sigma\left(\frac{h}{k}\right) \tag{8}$$

Now let us consider a distributed architecture. We show that in this case, due to the blocking technique, algorithm *TURBO* requires less communications than previously known optimal rank algorithms. Indeed, in order to achieve an optimal number of operations ($T_1 = O(h^\omega)$), those algorithms redesign the block structure of the matrix after each elimination step (k is modified). This redistribution involves $2m.2n$ communications at each step on a $2m \times 2n$ matrix; then, adding the pivot row communications, we obtain a total of $8mn$ communications. In particular, we consider the block row algorithm of Ibarra et al. [8]. Its communication volume for a $2m \times 2n$ matrix is denoted by $I(2m, 2n)$. Recall that it groups rows into two regions; now, the whole number

of communications performed is then

$$I(2m, 2n) = 2I(m, 2n) + 8mn \tag{9}$$

TURBO algorithm avoids such a redistribution. We denote our communication volume by $C(2m, 2n)$. This volume is a function of the five intermediate ranks (q, p, c, d, z, ranks of the matrices: U_1, V_2, C_3, D_3, Z_4, and $r = q+p+c+d+z$). Furthermore, each matrix X_i is computed with the *owner compute rule*: i.e. on the processor where it is supposed to be at the end. Now, the analysis of the dependency Graph 6 shows that each one of the following blocks must be communicated once: L_1, U_1, $B_{1(1..q,)}$, N_1, V_2, L_2, E_2, N_2, I_2, M_3, D_3, and F_3. This leads to the following volume*:

$$\begin{aligned} C(2m, 2n) = \\ 2C(m, n) + C(m-q, n-p) + C(m-p, n-q) + \\ C(m-q-c, n-q-d) + \\ mn + 2qm + 2pn + q^2 + pm + dn - pq - dq \end{aligned} \tag{10}$$

Thus, at the current step, the number of communications is $mn+2qm+2pn+q^2+pm+dn-pq-dq$; since p, q and d are smaller than $\min(m, n)$, this number is always less than the previous $8mn$. For our algorithm, the worst case occurs when the matrix is invertible. Then if the first two pivot blocks are of full rank ($p = q = \min(m, n)$ and $d = 0$), our number of communications is less than $6mn$ instead of $8mn$. In addition, and still in a full rank case, when the ranks are more evenly distributed on the blocks, say $p = q = d = c = \frac{\min(m,n)}{2}$, then our number of communications is less than $3.75mn$. We can therefore expect very good performance on the average.

However, solving the previous recurrence equations in the general case is quite complex. Therefore, in the following section, we compare those communications in practice, on specific dense and sparse matrices at a given step.

5 Practical communication performance

In this section, we compare the communication volumes between the row and block strategies. We first consider invertible matrices and then the general case.

* Remark that C_3 can be chosen instead of D_3 at step 3 of the algorithm. In that case, d must be replaced by c in the formula.

5.1 Invertible principal minors case

Consider a dense matrix. We study the differences in communication volume on a square $2n \times 2n$ invertible matrix, on P processors. There are then two possible cuttings. Suppose that the rows are cyclicly distributed on the P processors. At each step, the pivot row must be communicated to every other processor. This leads to the following volume of communications:

$$L(2n, 2n, P) = \sum_{k=1}^{2n} (P-1)(2n-k) = n(2n-1)(P-1) \tag{11}$$

Another way is to consider a cutting of the $2n \times 2n$ matrix into Q square blocks B_{ij} of size $\frac{2n}{\sqrt{Q}} \times \frac{2n}{\sqrt{Q}}$. Blocks are assumed cyclicly distributed on the processors. We also suppose here that there exists at least one invertible block at each step (*that* is very restrictive in exact computations). At step k, the block pivot row must be communicated, i.e. each one of the $\sqrt{Q} - k + 1$ blocks of this row (B_{kj} for j from k to $\sqrt{Q}$) must be sent to the $\sqrt{Q} - k$ other remaining blocks in its column. Then each remaining row performs the multiplication by the inverse of bloc B_{kk} and communicates the product $-B_{ik}.B_{kk}^{-1}$ to the $\sqrt{Q} - k$ other blocks of its row. Since 1 block over P is local, the volume of communications is then:

$$B(2n, 2n, P, Q) = \left(1 - \frac{1}{P}\right) \sum_{k=1}^{\sqrt{Q}} \left((\sqrt{Q} - k + 1)(\sqrt{Q} - k) \frac{2n}{\sqrt{Q}} \frac{2n}{\sqrt{Q}} + (\sqrt{Q} - k)^2 \frac{2n}{\sqrt{Q}} \frac{2n}{\sqrt{Q}} \right)$$

which gives

$$B(2n, 2n, P, Q) = \frac{2}{3}(2n)^2 \sqrt{Q} \left(1 - \frac{1}{P}\right) - \mathcal{O}(n^2) \tag{12}$$

Let ρ be the difference between the volume of monodimensional communications and the volume bidimensional communications divided by the volume of monodimensional communications. Then ρ measures the gain in communications between the two strategies. It is normalized in order to compare different kind of matrices.

Now, by taking $P = Q$, communications are reduced when using a block cutting. As a matter of fact, the gain between rows and blocks is as follows:

$$\rho = \frac{L(2n, 2n, P) - B(2n, 2n, P, P)}{L(2n, 2n, P)} \leq 1 - \frac{4}{3\sqrt{P}}. \tag{13}$$

Such a gain is then our goal for the singular case. We show in the next section that with our new algorithm, we can reach those performances *in general*.

5.2 General case

We now estimate the gain with our algorithm for a rectangular matrix $2m \times 2n$ of rank $r \leq min\{2m, 2n\}$. We then compute the communication volume, in the worst case, for only one phase (no recursion, only the steps previously shown), on 4 processors (one for each region). The cost function obtained with 4 processors is already quite complex:

$$\begin{aligned} C(2m, 2n, r, 4) = \\ mn + 2qm + 2pn + q^2 + pm + dn - pq - dq \end{aligned} \tag{14}$$

In order to give a more precise idea of the gain of our method, we compare this result to the volume of communications obtained by row. For the non invertible case, Formula 11 has to be modified as follows:

$$\begin{aligned} L(2n, r, P) = \\ \sum_{k=1}^{r}(P-1)(2n-k) = r(2n - \frac{r+1}{2})(P-1) \end{aligned} \tag{15}$$

where $r = q+p+c+d+z$ is the rank of the matrix. Next, Table 3 shows the gain obtained with the previously introduced matrices. The total effective communicated volumes of both (row and *TURBO*) methods are compared. These matrices are quite sparse. Unfortunately, the first version of our algorithm is implemented only for dense matrices. Still, we can see that our method is able to avoid some communications as soon as the matrices are not too special. In the table, the first three matrices have special rank conformations as seen in Table 1 ($d = z = 0$ for instance) and are very unbalanced (very small number of columns compared to the number of rows): in that case a row method can be much more efficient since it can communicate only the smallest dimension. However, in all the other cases we are able to achieve very good performances: for the less rectangular matrices, we have a gain ρ very close to the aimed one (Equation 13 gives $1 - \frac{4}{3\sqrt{4}} = \frac{1}{3} \approx 33\%$).

Now the problem is that the recursive setting is rather delicate. Indeed, to limit the structure overhead, the recursive cutting threshold must be rather high. The induced parallelism is thus not so extensible and the algorithm is interesting on a relatively small number of processors (4, 8, 16, . . .). Nevertheless, there is the possibility of using this cutting only for the first stages (greediest in communications) and then of switching to the row algorithm, for instance.

Moreover, this algorithm can be easily adapted to take advantage of sparse

Matrix	$2m \times 2n$	r	$\rho = \frac{L-C}{L}$
ch5-5.b2	600x200	176	-57.97%
mk9.b2	1260x378	343	-67.36%
ch6-6.b2	2400x450	415	-123.66%
ch4-4.b2	96x72	57	10.40%
ch5-5.b3	600x600	424	32.80%
mk9.b3	945x1260	875	11.80%
robot24_m5	404x302	262	9.08%
rkat7_m5	694x738	611	34.02%
f855_m9	2456x2511	2331	34.68%
cyclic8_m11	4562x5761	3903	21.02%

Table 3
Communication volume gain

matrices. With the only restriction that in this case, it is not possible to completely apply reordering heuristics to the whole matrix without adding communications (it is anyway feasible to apply them locally, in each block, but only with limited effects [4]).

6 Conclusions

To conclude, we developed a new block TU elimination algorithm. Its theoretical sequential and parallel arithmetic complexities are similar to those of the most efficient current elimination algorithms for this problem. Besides, it is particularly adapted to the singular matrices and makes it possible to compute the rank in an exact way. Furthermore, it allows a more flexible management of the scheduling (adaptive grain) and avoids a third of the communications when used with only one level of recursion on 4 processors.

In addition, if the increase in locality reduces the number of communications, it also makes it possible to increase the speed by a greater benefit of the cache effects. There remains to implement efficiently the parallel finite field subroutines to test the effectiveness in terms of speed-up. For instance, recent sequential experiments ([14]) show that classical Gaussian elimination can be performed at a speed of about 40 Million of field operations per second (MFop/s) on a pentium III 735 MHz, whereas fast matrix multiplication can achieve a speed close to 600 MFop/s on the same machine. When considering that Gaussian elimination requires $\frac{2}{3}n^3$ operations, it means that our algorithm

requires at most seven times that number of operations. The compared speeds shows that even a sequential speed-up can be achieved.

Lastly, there remains also to study an effective method to reorder sparse matrices effectively in parallel. Indeed, even though designed for dense matrices, our algorithm can also be used on sparse matrices. But, in order to attain high speeds, we need to gather the non-zero elements. Figure 8 shows the result of an iteration of our algorithm on a sparse matrix (Figure 7) arising in the computation Gröbner bases.

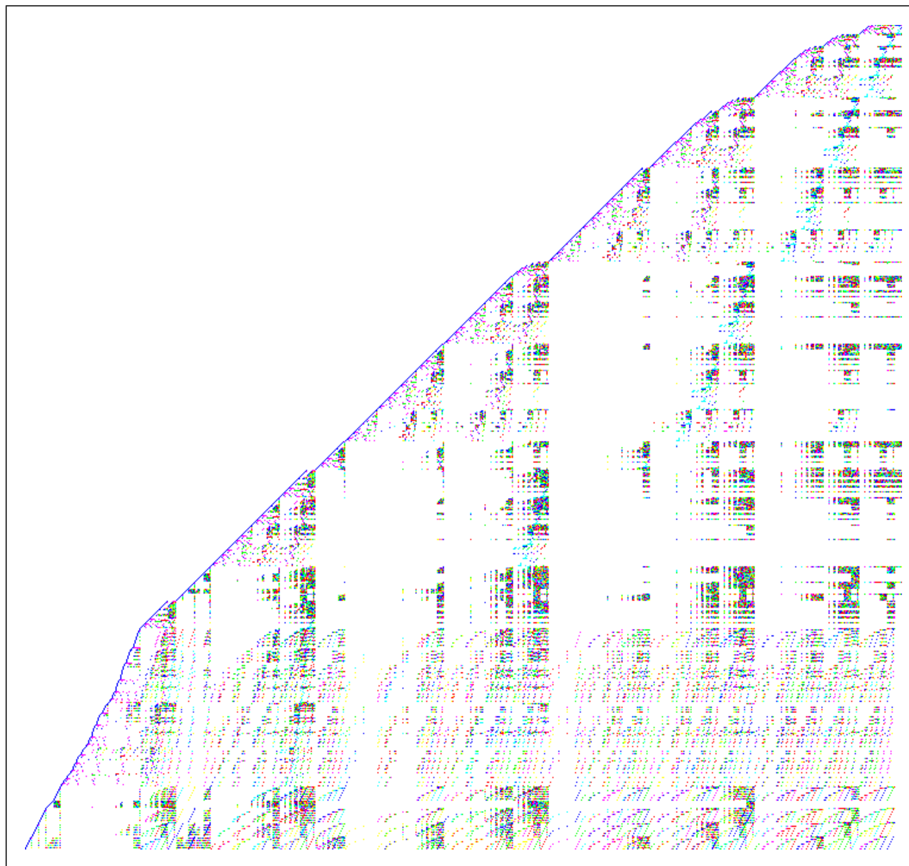

Fig. 7. Matrix rkat7_mat5, 694×738 of rank 611

We see that a rather significant fill-in occurs during the last phases of this iteration: the final U shape of this matrix has 105516 non zero elements. By way of comparison, the final LU shape of this matrix, computed with a row algorithm, has 64622 non zero elements. Moreover, using a reordering technique, one can obtain a triangular form containing much fewer non zero elements (39477 for instance for this matrix [4, section 5.4.5]). Therefore, designing an efficient *block reordering* technique seems to be an important open question.

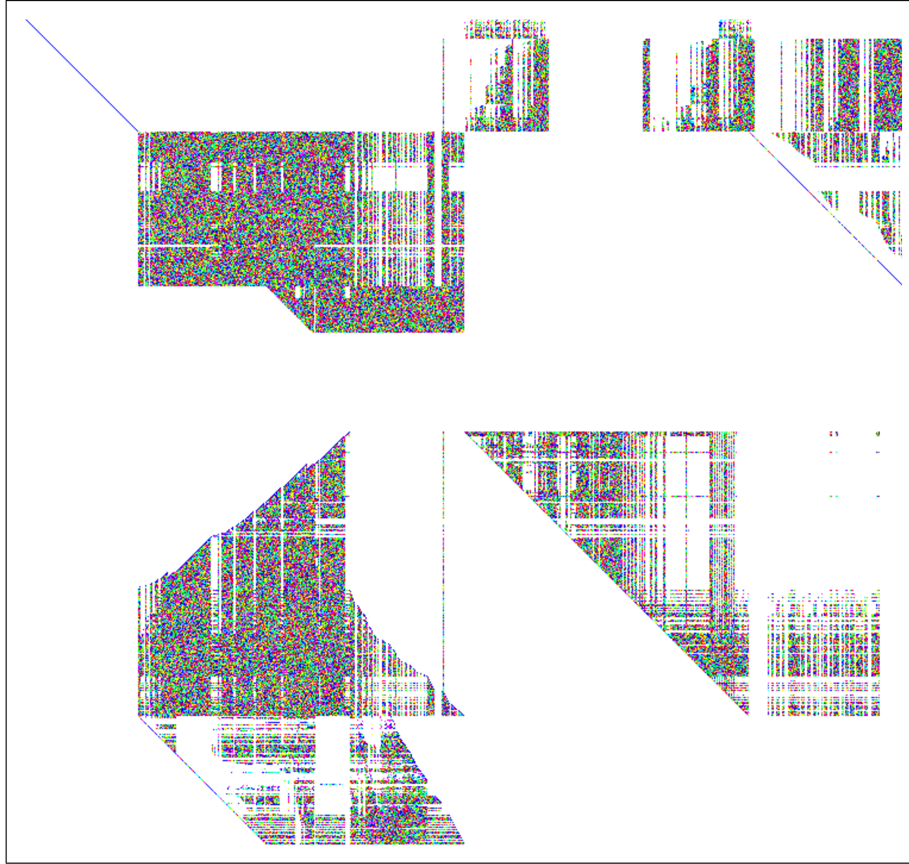

Fig. 8. Matrix rkat7_mat5 after one phase of our algorithm

References

[1] B. Buchberger, Gröbner bases: An algorithmic method in polynomial ideal theory, in: N. K. Bose (Ed.), Recent Trends in Multidimensional Systems Theory, Mathematics and its applications, D. Reidel Publishing Company, Dordrecht, The Netherlands, 1985, Ch. 6, pp. 184–232.

[2] J.-C. Faugère, Parallelization of Gröbner basis, in: H. Hong (Ed.), First International Symposium on Parallel Symbolic Computation, PASCO '94, Hagenberg/Linz, Austria, Vol. 5 of Lecture notes series in computing, 1994, pp. 124–132.

[3] W. D. Wallis, A. P. Street, J. S. Wallis, Combinatorics: Room Squares, Sum-Free Sets, Hadamard Matrices, Vol. 292 of Lecture Notes in Mathematics, Springer-Verlag, Berlin, 1972.

[4] J.-G. Dumas, Algorithmes parallèles efficaces pour le calcul formel : algèbre linéaire creuse et extensions algébriques, Ph.D. thesis, Institut National Polytechnique de Grenoble, France, `ftp://ftp.imag.fr/pub/-Mediatheque.IMAG/theses/2000/Dumas.Jean-Guillaume` (Dec. 2000).

[5] V. Kumar, A. Grama, A. Gupta, G. Karypis, Introduction to parallel computing. Design and analysis of algorithms, The Benjamin/Cummings Publishing Company, Inc., 1994.

[6] J. J. Dongarra, J. D. Croz, S. Hammarling, I. Duff, A set of level 3 Basic Linear Algebra Subprograms, Transactions on Mathematical Software 16 (1) (1990) 1–17, www.acm.org/pubs/toc/Abstracts/0098-3500/79170.html.

[7] O. H. Ibarra, S. Moran, L. E. Rosier, A note on the parallel complexity of computing the rank of order n matrices, Information Processing Letters 11 (4–5) (1980) 162–162.

[8] O. H. Ibarra, S. Moran, R. Hui, A generalization of the fast LUP matrix decomposition algorithm and applications, Journal of Algorithms 3 (1) (1982) 45–56.

[9] A. Storjohann, Algorithms for matrix canonical forms, Ph.D. thesis, Department of Computer Science, Swiss Federal Institute of Technology—ETH Zurich (Dec. 2000).

[10] D. Bini, V. Pan, Polynomial and Matrix Computations, Volume 1: Fundamental Algorithms., Birkhauser, Boston, 1994.

[11] J.-G. Dumas, J.-L. Roch, A fast parallel block algorithm for exact triangularization of rectangular matrices, in: SPAA'01. Proceedings of the Thirteenth ACM Symposium on Parallel Algorithms and Architectures, Kreta, Greece., 2001, pp. 324–325.

[12] A. Björner, V. Welker, Complexes of directed graphs, SIAM Journal on Discrete Mathematics 12 (4) (1999) 413–424.
URL http://epubs.siam.org/sam-bin/dbq/article/33872

[13] V. Reiner, J. Roberts, Minimal resolutions and the homology of matching and chessboard complexes, Journal of Algebraic Combinatorics 11 (2) (2000) 135–154.

[14] J.-G. Dumas, T. Gautier, C. Pernet, Finite fields linear algebra subroutines, in: T. Mora (Ed.), Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation, Lille, France, ACM Press, New York, 2002.

[15] J.-C. Faugère, A new efficient algorithm for computing Gröbner bases (F_4), Tech. rep., Laboratoire d'Informatique de Paris 6, http://www-calfor.lip6.fr/~jcf (Jan. 1999).

[16] A. Björner, L. Lovász, S. T. Vrećica, R. T. Živaljević, Chessboard complexes and matching complexes., Journal of the London Mathematical Society 49 (1) (1994) 25–39.

[17] J.-G. Dumas, B. D. Saunders, G. Villard, Integer Smith form via the Valence: experience with large sparse matrices from Homology, in: C. Traverso (Ed.), Proceedings of the 2000 International Symposium on Symbolic and Algebraic Computation, Saint Andrews, Scotland, ACM Press, New York, 2000, pp. 95–105.

[18] V. Strassen, Gaussian elimination is not optimal, Numerische Mathematik 13 (1969) 354–356.

[19] D. Coppersmith, S. Winograd, Matrix multiplication via arithmetic progressions, Journal of Symbolic Computation 9 (3) (1990) 251–280.

[20] A. V. Aho, J. E. Hopcroft, J. D. Ullman, The Design and Analysis of

Computer Algorithms, Addison-Wesley, 1974.
[21] K. Mulmuley, A fast parallel algorithm to compute the rank of a matrix, Combinatorica 7 (1) (1987) 101–104.
[22] J. JáJá, Introduction to Parallel Algorithms, Addison-Wesley, New York, 1992.
[23] G. G. H. Cavalheiro, M. Doreille, F. Galilée, J.-L. Roch, Athapascan-1: On-line building data flow graph in a parallel language, in: PACT'98: Proceedings of the International Conference on Parallel Architectures and Compilation Techniques, Paris, France, 1998.
[24] R. L. Graham, Bounds on certain multiprocessing timing anomalies, SIAM Journal of Applied Mathematics 17 (2) (1969) 416–429.