A new algorithm for computing the Smith normal form and its implementation on parallel machines

Gerold Jäger
Mathematisches Seminar
Christian-Albrechts-Universität zu Kiel
Christian-Albrechts-Platz 4
24118 Kiel
Germany
gej@numerik.uni-kiel.de

Abstract

Smith normal form computation is important in many topics, e.g. group theory and number theory. For matrices over the rings $\mathbb{Z}$ and $\mathbb{F}_2[x]$, we introduce a new Smith normal form algorithm, called Triangular Band Matrix Algorithm, which first computes the Hermite Normalform and then step by step the diagonal form and the Smith normal form. In comparison to the Kannan Bachem Algorithm, which computes the Smith normal form by alternately computing the Hermite normal form and the left Hermite normal form, the theoretical advantage is, that we only once apply the expensive Hermite normal form step. We parallelize the Triangular Band Matrix Algorithm and get a better complexity analysis than for previous parallel algorithms, like the Kannan Bachem Algorithm and the Hartley Hawkes Algorithm. In the part, which is different to the Kannan Bachem Algorithm, the Triangular Band Matrix Algorithm leads to a better efficiency and smaller execution times, even for large example matrices.

1 Introduction

A matrix in $R^{m,n}$ over a Euclidean ring R with rank r is in *Smith normal form*, if it is a diagonal matrix with the first r diagonal elements being divisors of the next diagonal element and the last diagonal elements being zero. A theorem of Smith [13] says that you can obtain from an arbitrary matrix in $R^{m,n}$ the uniquely determined Smith normal form by doing unimodular row and column operations.
There are many algorithms for efficiently computing the Smith normal form, most of them only for one of the rings $\mathbb{Z}$ or $\mathbb{F}[x]$. Some of these algorithms are probabilistic ([1] for $R = \mathbb{Z}$, [11] for $R = \mathbb{F}[x]$, [17] for $R = \mathbb{Q}[x]$). Deterministic algorithms for $R = \mathbb{Z}$ often use modular techniques ([2], [4], [12, Chapter 8.4], [14], [15], [16]). The disadvantage of these algorithms is that they are unable to compute the transformation matrices. In [11] an algorithm for $R = \mathbb{F}[x]$ and in [14] an algorithm for $\mathbb{Z}$ is introduced, which compute the transformation matrices. The algorithm of Storjohann, which uses modular techniques and computes the transformation matrices after the computation of the Smith normal form, gives the best known complexity analysis for Smith normal form algorithms over $\mathbb{Z}$. The only algorithms, which work for both the ring $\mathbb{Z}$ and the ring $\mathbb{F}[x]$ and which compute the transformation matrices during the computation of the Smith normal form, are the algorithm of Kannan, Bachem [9] and the algorithm of Hartley, Hawkes [3]. These algorithms compute the diagonal form first and finally with an elementary algorithm the Smith normal form. The algorithm of Kannan, Bachem alternately computes the Hermite normal form and the transposed left Hermite normal form until the matrix is in diagonal form. The theoretical problem of this algorithm is that the number of Hermite normal form and left Hermite normal form computations can be very large. The algorithm of Hartley, Hawkes alternately computes the gcd of a whole row and the gcd of a whole column and replaces this row and column by this gcd and zeros. In this way you can enlarge the part of the matrix, which is in diagonal form, step by step.

In Section 3 we describe our new algorithm. It begins with a Hermite normal form computation like the algorithm of Kannan, Bachem, then transforms the matrix into a triangular band matrix, i.e. a matrix with a main

diagonal and a side diagonal above it. Finally we transform this matrix into a diagonal form so that we can use the mentioned elementary algorithm to get the Smith normal form. We call this algorithm *Triangular Band Matrix Algorithm*. In contrast to the Kannan Bachem Algorithm, the Hermite normal form computation is executed only once.

For being able to compute larger matrices in less time it is reasonable to parallelize the algorithms. In [7], [8] and [18] parallel probabilistic algorithms are introduced for the ring $\mathbb{F}[x]$, but without experimental results. In [6] the algorithms of Kannan, Bachem and Hartley, Hawkes are parallelized for the rings $\mathbb{Z}$ and $\mathbb{F}[x]$, including complexity results and experiments. In Section 4 we parallelize the Triangular Band Matrix Algorithm and compare it with these algorithms. Like for the Kannan Bachem Algorithm and the Hartley Hawkes Algorithm, row distribution of the matrix is better, if we mainly use column operations. Analogously column distribution is better, if we mainly use row operations. As only once we use the expensive algorithm, which transforms a row distributed matrix into a column distributed one, we can show that the complexity of the communication operations of the Triangular Band Matrix Algorithm is better than those of the Kannan Bachem Algorithm and the Hartley Hawkes Algorithm.

In Section 5 we test the parallel versions of the algorithms. The execution time of the Triangular Band Matrix Algorithm is dominated by the Hermite normal form part and so behaves similarly to the Kannan Bachem Algorithm. If we compare the Triangular Band Matrix Algorithm with the Kannan Bachem Algorithm, the Triangular Band Matrix Algorithm produces better results. The parallel version is efficient, i.e. if we double the number of processes, the execution time is nearly halved.

2 Preliminaries

Let R be a commutative, integral ring with 1. Let R be *Euclidean*, i.e. there is a mapping $\phi : R \setminus \{0\} \to \mathbb{N}_0$ such that for $a \in R, b \in R \setminus \{0\}$ exist $q, r \in R$ with $a = qb + r$ and $r = 0 \vee \phi(r) < \phi(b)$. We consider the following two examples:

i) For the set $\mathbb{Z}$ of integers we choose $\phi := |\cdot|$ and $\mathcal{R} := \mathbb{N}_0$. For $a \in \mathbb{R}$ let $\lfloor a \rfloor$ be the largest integer $\leq a$. We define $q := \lfloor a/b \rfloor, r := a - \lfloor a/b \rfloor \cdot b$.

ii) For the polynomial ring $\mathbb{F}[x]$ with a field $\mathbb{F}$ we choose $\phi := \deg$ and $\mathcal{R} := \{\text{Monic polynomials over } \mathbb{F}[x]\}$. q and r are uniquely determined by polynomial division. For $A \in \mathbb{F}[x]^{m,n}$ let $\lceil A \rceil_{\deg} := \max_{1 \leq i,j \leq n}\{\deg(A_{i,j})\}$.

For $a \in R, b \in R \setminus \{0\}$ we define $\psi_1(a,b) := q$ and $\psi_2(a,b) := r$. For $d \in R$ we denote by $\mu(d)$ the number of prime factors of d with multiplicity.

Definition 1 *$GL_n(R)$ is the group of matrices in $R^{n,n}$ whose determinant is a unit in the ring R. These matrices are called unimodular matrices.*

Definition 2 *$A \in R^{m,n}$ with rank r is in Hermite normal form (HNF), if:*

a) *$\exists i_1, \cdots, i_r$ with $1 \leq i_1 < \cdots < i_r \leq m$ with $A_{i_j,j} \in \mathcal{R} \setminus 0$ for $1 \leq j \leq r$.*

b) *$A_{i,j} = 0$ for $1 \leq i \leq i_j - 1,\ 1 \leq j \leq r$.*

c) *The columns $r+1, \cdots, n$ are zero.*

d) *$A_{i_j,l} = \psi_2(A_{i_j,l}, A_{i_j,j})$ for $1 \leq l < j \leq r$.*

A is in left Hermite normal form (LHNF), if its transpose A^T is in HNF.

Theorem 1 [5] *Let $A \in R^{m,n}$. There exists a matrix $V \in GL_n(R)$ such that $H = AV$ is in HNF. The matrix H is uniquely determined.*

Definition 3 *$A \in R^{m,n}$ with rank r is in Smith normal form (SNF), if:*

a) *A is a diagonal matrix.*

b) *$A_{i,i} \in \mathcal{R} \setminus \{0\}$ for $1 \leq i \leq r$.*

c) *$A_{i,i} \mid A_{i+1,i+1}$ for $1 \leq i \leq r-1$.*

d) *$A_{i,i} = 0$ for $r+1 \leq i \leq \min\{m,n\}$.*

Theorem 2 [13] *Let $A \in R^{m,n}$. There exist matrices $U \in GL_m(R)$ and $V \in GL_n(R)$ such that $C = UAV$ is in SNF. The matrix C is uniquely determined. (U, V are called the left hand and the right hand transformation matrix.)*

3 Triangular Band Matrix Algorithm

3.1 Auxiliary algorithm BANDTODIAG

If a square matrix with full rank consists of a main diagonal with a side diagonal underneath, we call it a lower triangular band matrix. If a square matrix with full rank consists of a main diagonal and a side diagonal above it, we call it an upper triangular band matrix. As our new algorithm computes from the Hermite normal form an upper triangular band matrix, then a diagonal form and finally with an elementary algorithm [10, p. 318, Algorithmus 11.5.15] the Smith normal form, we call it *Triangular Band Matrix Algorithm*. The following auxiliary algorithm BANDTODIAG computes from an upper triangular band matrix a diagonal matrix. In this algorithm we use the procedures **ROW-ONE-GCD** and **COL-ONE-GCD**. For i, j, l with $1 \leq i \leq m,\ 1 \leq j < l \leq n$ ROW-ONE-GCD (A, i, j, l)

transforms A so that $A_{i,j}^{\text{new}} = \gcd(A_{i,j}^{\text{old}}, A_{i,l}^{\text{old}}) \in \mathcal{R}, A_{i,l}^{\text{new}} = 0$. This is done by applying the following lemma:

Lemma 1 [10, p. 313, Hilfssatz 11.5.9] *Let $a, b \in R$ be not both zero. Let $d = gcd(a,b) \in \mathcal{R}$, $u, v \in R$ with $d = au + bv, s = \frac{a}{d}$, $t = \frac{b}{d}$ and $V = \begin{pmatrix} u & -t \\ v & s \end{pmatrix}$. It holds: $V \in GL_2(R)$ and $(a,\ b)\,V = (d,\ 0)$.*

The procedure COL-ONE-GCD is analogously defined with the role of rows and columns exchanged.

Algorithm 1 *(BANDTODIAG)*

INPUT $A \in R^{r,r}$ upper triangular band matrix with rank r
1 *WHILE (A is not in diagonal form)*
2 *FOR $l = 1, \cdots, r-1$*
3 *ROW-ONE-GCD$(A, l, l, l+1)$*
4 *FOR $l = 1, \cdots, r-1$*
5 *COL-ONE-GCD$(A, l, l, l+1)$*
OUTPUT $A =$ BANDTODIAG$(A) \in R^{r,r}$ in diagonal form

Correctness of Algorithm 1:
Assertion: a) The FOR loop in step 2 transforms an upper triangular band matrix into a lower triangular band matrix and the number of side diagonal elements does not increase.
b) The FOR loop in step 4 transforms a lower triangular band matrix into an upper triangular band matrix and the number of side diagonal elements does not increase.
Proof: a) Inductively we get that starting step 3 with l, the columns l and $l+1$ are zero in all entries except for l and $l+1$:

$$\begin{array}{c} \\ \\ \\ \\ l \\ l+1 \\ \\ \\ \\ \end{array} \begin{pmatrix} * & 0 & \cdots & & & & & 0 \\ * & * & \ddots & & & & & \vdots \\ 0 & \ddots & \ddots & 0 & 0 & & & \\ \vdots & \ddots & * & * & * & 0 & & \\ & & 0 & 0 & * & \ddots & \ddots & \\ & & \vdots & \vdots & \ddots & \ddots & \ddots & 0 \\ & & & & & & \ddots & * \\ 0 & \cdots & 0 & 0 & \cdots & & 0 & * \end{pmatrix}$$
$$\begin{array}{cc} l & l+1 \end{array}$$

Because of the definition of ROW-ONE-GCD and Lemma 1, step 3 implies:

$$\begin{pmatrix} A_{l,l} & 0 \\ A_{l+1,l} & A_{l+1,l+1} \end{pmatrix} = \begin{pmatrix} A_{l,l} & A_{l,l+1} \\ 0 & A_{l+1,l+1} \end{pmatrix} \begin{pmatrix} u & -A_{l,l+1}/d \\ v & A_{l,l}/d \end{pmatrix} \quad (1)$$

So an entry from the upper diagonal is transformed into the lower diagonal, and the number of side diagonal elements does not increase. If $A_{l,l+1}^{\text{old}} = 0$, we choose $u = e, v = 0$ with a unit e, and so the entries $A_{l+1,l}$ and $A_{l,l+1}$ stay zero, and the number of side diagonal elements does not change.
b) Analogous to a). ◊
It remains to show that Algorithm 1 terminates, i.e. the number of side diagonal elements becomes zero in finite time.
If after completion of one of the two FOR loops in step 2 and 4, respectively, we have for $l \in \{1, \cdots, r-1\}$: $A_{l+1,l} = 0$ and $A_{l,l+1} = 0$, then this characteristic stays true in the whole WHILE loop.
We consider the entrance into step 3 and distinguish two cases:
Case 1: $A_{l,l}^{\text{old}} \mid A_{l,l+1}^{\text{old}}$
We choose $u = e, v = 0$ with a unit e and obtain $A_{l+1,l}^{\text{new}} = 0$ after step 3 because of (1) and $v = 0$. So the number of side diagonal elements decreases by 1.
Case 2: $A_{l,l}^{\text{old}} \nmid A_{l,l+1}^{\text{old}}$
After step 3 we have: $A_{l,l}^{\text{new}} = \gcd(A_{l,l}^{\text{old}}, A_{l,l+1}^{\text{old}})$. So $\mu(A_{l,l})$ decreases at least by 1. As the product of the diagonal elements keeps constant, $\mu(A_{l+1,l+1})$ increases at least by 1.
While executing step 3 either a side diagonal element becomes zero or at least one prime factor is pushed downwards by a diagonal element. We can show that analogously for step 5.
Let d be the product of the elementary divisors of A. As the determinant of A is $e \cdot d$ with a unit e, the calls of the steps 3 and 5 for $R = \mathbb{Z}$ can be estimated by:

$$\begin{aligned} (r-1)\log_2(d) + r - 1 &\leq r\log_2(r! \cdot \|A\|_\infty^r) + r - 1 \\ &\leq 2r^2 \log_2(r \cdot \|A\|_\infty) \end{aligned}$$

For $R = \mathbb{F}[x]$ the calls can be estimated by:

$$\begin{aligned} (r-1)\deg d + r - 1 &\leq r^2 \lceil A \rceil_{\deg} + r - 1 \\ &\leq 2r^2 \lceil A \rceil_{\deg} \end{aligned}$$

So Algorithm 1 terminates. □
From the proof of correctness it follows:

Remark 1 *Let $A \in R^{r,r}$ have full rank. Then the steps 3 and 5 of Algorithm 1 can be executed for $R = \mathbb{Z}$ $O(r^2 \log_2(r\|A\|_\infty))$ times and for $R = \mathbb{F}[x]$ $O(r^2 \lceil A \rceil_{\deg})$ times.*

3.2 Main algorithm

Algorithm 2 *(Triangular Band Matrix Algorithm)*

INPUT $A = [a'_1, \cdots, a'_m]^T = [a_1, \cdots, a_n] \in R^{m,n}$
1 $A = HNF(A)$ *with rank* r *and row indices* $i_1, \cdots, i_r$
2 *IF* $r < m$
3 *THEN FOR* $l = 1, \cdots, r$
4 *IF* $l \neq i_l$
5 *THEN* $a'_l \leftrightarrow a'_{i_l}$
6 *FOR* $l = r, \cdots, 1$ *(backwards)*
7 *FOR* $s = r+1, \cdots, m$
8 *COL-ONE-GCD*(A, l, l, s)
9 *FOR* $l = 1, \cdots, r-1$
10 *FOR* $s = r-1, \cdots, l+1$ *(backwards)*
11 *COL-ONE-GCD*$(A, s, l, s+1)$
12 *ROW-ONE-GCD*$(A, s, s, s+1)$
13 *COL-ONE-GCD*$(A, l, l, l+1)$
14 $A = BANDTODIAG(A)$
15 $A = DIAGTOSMITH(A)$
OUTPUT $A = SNF(A) \in R^{m,n}$

Correctness of Algorithm 2: During the steps 3 to 5 the rows are exchanged, so that $(A_{i,j})_{1 \leq i,j \leq r}$ is in HNF and has rank r. During the steps 6 to 8 $(A_{i,j})_{r+1 \leq i \leq m, 1 \leq j \leq r}$ is gradually transformed to a zero matrix by transforming for $l = r, \cdots, 1$ and $s = $ r+1, $\cdots, m$ the entry $A_{s,l}$ to zero. After step 8 all entries outside of $(A_{i,j})_{1 \leq i,j \leq r}$ are zero. In the following we only consider this submatrix.
During the steps 9 to 13 the matrix A is transformed to an upper triangular band matrix, which easily follows from the following assertion. During step 14 the matrix is transformed to diagonal form and during step 15 to SNF. □

Assertion: After the FOR loop has executed step 9 with l, $(A_{i,j})_{1 \leq i,j \leq l+1}$ is an upper triangular band matrix and $(A_{i,j})_{l+1 \leq i,j \leq r}$ is a lower triangular matrix and the remaining entries are zero (see Figure 1).

Sketch of the proof of the assertion (for a more detailed proof we refer to the full version):
Induction on l.
$\underline{l = 0}$: Before the FOR loop has executed step 9 with 1, the assertion holds, as the whole matrix is a lower triangular matrix.
$\underline{l \to l+1}$: We assume that the FOR loop of step 9 has passed l and has the form (2). By the steps 11 and 12, the entry $A_{s+1,l+1}$ is transformed to zero. By step 13, the entry $A_{l+1,l}$ is transformed to zero, while the entry $A_{l,l+1}$ might be transformed to a non-zero element. So we have the desired form of Figure 1 for $l+1$. ◊

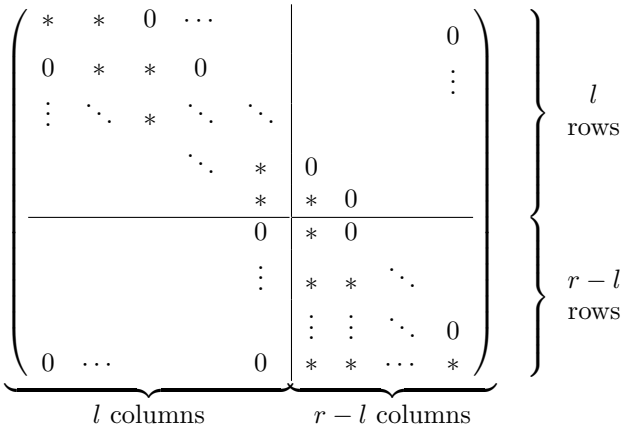

Figure 1. Matrix from the assertion

4 Parallel Triangular Band Matrix Algorithm

4.1 Idea of Parallelization

A parallel program is a set of independent processes with data being interchanged between the processes. For interchanging the data we need the following procedures:

- With **BROADCAST** x a process sends a variable x to all other processes.
- With **BROAD-RECEIVE** x **FROM z** a process receives a variable x from the process with the number z which it has sent with BROADCAST.
- With **SEND** x **TO z** a process sends a variable x to the process with the number z.
- With **SEND-RECEIVE** x **FROM z** a process receives a variable x from the process with the number z which it has sent with SEND.

The matrix whose Smith normal form shall be computed has to be distributed to the different processes as uniformly as possible. Like in [6], we distribute rows, if mainly column operations are used, and columns, if mainly row operations are used. We will also switch between both (see the procedure PAR-ROWTOCOL). We consider a row distribution. Let the matrix $\bar{A} \in R^{m,n}$ be distributed on q processes and let the z-th process consist of $k(z)$ rows. Every process has as input a matrix $A \in R^{k(z),n}$, where $\sum_{z=1}^{q} k(z) = m$. For every process let the order of the rows be equal to the original order. At any time we can obtain the complete

matrix by putting together these matrices. We choose the following distribution:
The process with the number z receives the rows $z, z+q, \cdots, z+\lfloor (m-z)/q \rfloor \cdot q$.
Further we need the following functions:

- For $1 \leq l \leq m$, **ROW-TASK** (l) returns the number of the process owning the l-th row.
- For $1 \leq l \leq m$, **ROW-NUM** (l) returns the row number of the original l-th row, if the row lies on its own process, and otherwise the row number, which it would get, if it would be inserted into its own process.

Analogously we define the column distribution and the functions **COL-TASK** and **COL-NUM**. Since only column operations are performed on the right hand transformation matrix V, we choose for V a row distribution. For the left hand transformation matrix U we choose a column distribution.

Theorem 3 [19] *Let $p = \max\{m, n\}$. There is a parallel HNF algorithm in which $O(p^2)$ BROADCAST operations are performed and $O(p^3)$ ring elements are sent with BROADCAST.*

4.2 Auxiliary algorithm PAR-DIAGTOSMITH

The auxiliary algorithm DIAGTOSMITH is parallelized trivially. Every diagonal element is broadcast from the process it lies on to all other processes so that the operations can be performed. In this algorithm $O(p^2)$ BROADCAST operations are performed and $O(p^2)$ ring elements are sent with BROADCAST. We implement this algorithm for column distribution.

4.3 Auxiliary algorithm PAR-ROWTOCOL

This algorithm changes a row distributed matrix into a column distributed matrix. Let $A \in R^{k(z),n}$ with $\sum_{z=1}^{q} k(z) = m$ be row distributed. A is transformed into a matrix $B \in R^{m,k'(z)}$ with $\sum_{z=1}^{q} k'(z) = n$. Every process communicates with every other process. Let $x \neq y$ be two processes. Then the SEND operation from x to y has the following form:

$$x : \text{SEND}\,\{A_{s,t} \mid 1 \leq s \leq k(x),\ 1 \leq t \leq n,\ y = \text{COL-TASK}\,(t)\} \ \rightarrow\ y$$

The RECEIVE operation y from x has the following form:

$$x \rightarrow y : \text{RECEIVE}\,\{B_{s,t} \mid 1 \leq s \leq m,\ 1 \leq t \leq k'(y),\ x = \text{ROW-TASK}\,(s)\}$$

This algorithm does not need to be applied to the transformation matrices, as we always transform the left hand transformation matrix U by row operations and the right hand transformation matrix V by column operations.

4.4 Main algorithm

In comparison to the original Algorithm 2, in the parallel Triangular Band Matrix Algorithm the steps 11 to 28 are added. These steps are not necessary for the correctness. The steps 11 to 17 are needed for the case that a diagonal element is a divisor of all elements in the same column, and in this case all elements in this column become zero (we have this case very often in practice). The steps 18 to 28 reduce the non-diagonal elements so that the entries do not explode.
Since after step 1 of Algorithm 2 the most operations are row operations, we change to column distribution after the HNF algorithm. So we implement the parallel versions **PAR-ROW-ONE-GCD** and **PAR-COL-ONE-GCD** of the procedures ROW-ONE-GCD and COL-ONE-GCD for a column distribution. The procedure PAR-ROW-ONE-GCD is slower than PAR-COL-ONE-GCD, as column operations are performed on a column distributed matrix and so two instead of one BROADCAST operations are needed. The parallel version **PAR-BANDTODIAG** results from the algorithm BANDTODIAG by replacing ROW-ONE-GCD by PAR-ROW-ONE-GCD and COL-ONE-GCD by PAR-COL-ONE-GCD.

Algorithm 3 *(Parallel Triangular Band Matrix Algorithm)*

INPUT Number of processes q, number of its own process z, $A \in R^{k(z),n}$ and $\sum_{z=1}^{q} k(z) = m$

1 $A = PAR\text{-}HNF(A)$ *with rank r and row indices* $i_1, \cdots, i_r$
2 $B = PAR\text{-}ROWTOCOL\,(A)$ $(B = [b'_1, \cdots, b'_m]^T = [b_1, \cdots, b_{k'(z)}] \in R^{m,k'(z)}$ *and* $\sum_{z=1}^{q} k'(z) = n)$
3 *IF* $r < m$
4 *THEN FOR* $l = 1, \cdots, r$
5 *IF* $l \neq i_l$
6 *THEN* $b'_l \leftrightarrow b'_{i_l}$
7 *FOR* $l = r, \cdots, 1$ *(backwards)*
8 *FOR* $s = r+1, \cdots, m$
9 $PAR\text{-}COL\text{-}ONE\text{-}GCD\,(B, l, l, s)$
10 *FOR* $l = 1, \cdots, r-1$
11 $y = COL\text{-}TASK\,(l)$
12 *IF* $y = z$
13 *THEN* $h = COL\text{-}NUM\,(l)$
14 *BROADCAST* b_h
15 *FOR* $t = l+1, \cdots, r$
16 $B_{t,h} = \psi_2(B_{t,h}, B_{l,h})$
(orig.: $B_{t,l} = \psi_2(B_{t,l}, B_{l,l})$ *)*

17 *ELSE BROAD-RECEIVE v FROM y*
18 *FOR* $s = r-1, \cdots, l$ *(backwards)*
19 $y = COL\text{-}TASK(s)$
20 $h = COL\text{-}NUM(s)$
21 *IF* $y = z$
22 *THEN BROADCAST* b_h
23 *ELSE BROAD-RECEIVE v FROM y*
24 *Insert v as h-th col. vector of B*
25 *FOR* $t = s+1, \cdots, r$
26 $b'_t = b'_t - \psi_1(B_{t,h}, B_{s,h})b'_s$
(orig.: $b'_t = b'_t - \psi_1(B_{t,s}, B_{s,s})b'_s$*)*
27 *IF NOT* $y = z$
28 *THEN Remove v as h-th col. vector of B*
29 *FOR* $s = r-1, \cdots, l+1$ *(backwards)*
30 $PAR\text{-}COL\text{-}ONE\text{-}GCD(B, s, l, s+1)$
31 $PAR\text{-}ROW\text{-}ONE\text{-}GCD(B, s, s, s+1)$
32 $PAR\text{-}COL\text{-}ONE\text{-}GCD(B, l, l, l+1)$
33 $B = PAR\text{-}BANDTODIAG(B)$
34 $B = PAR\text{-}DIAGTOSMITH(B)$
$OUTPUT\ B = PAR\text{-}SNF(A) \in R^{m,k'(z)}$

Theorem 4 *Let* $\bar{A} \in R^{m,n}$ *with* $p = \max\{m,n\}$*. For* $R = \mathbb{Z}$ *and* $R = \mathbb{F}[x]$*, respectively, we have:*
a) *In the steps 1 to 32 and 34 of Algorithm 3* $O(p^2)$ *BROADCAST operations are performed and* $O(p^3)$ *ring elements are sent with BROADCAST. Further* $O(q^2)$ *SEND operations are performed and* $O(p^2)$ *ring elements are sent with SEND.*
b) *In step 33* $O(p^2 \log_2(p\|\bar{A}\|_\infty))$ *and* $O(p^2\lceil\bar{A}\rceil_{\deg})$ *BROADCAST operations are performed, respectively, and* $O(p^2 \log_2(p\|\bar{A}\|_\infty))$ *and* $O(p^2\lceil\bar{A}\rceil_{\deg})$ *ring elements are sent with BROADCAST, respectively.*

Proof: a) Results from the fact that all BROADCAST operations are in at most two FOR loops and at most p ring elements are sent.
b) Follows from Remark 1 and since the procedures PAR-ROW-ONE-GCD and PAR-COL-ONE-GCD in algorithm PAR-BANDTODIAG are performed on (2×2) matrices. □

As we should choose the number of processes smaller than the row or column number, we obtain that apart from step 33 which is very fast in practice, only the row or column number comes in as a square in the number of parallel operations.
For comparison the results for the parallel Kannan Bachem Algorithm and the parallel Hartley Hawkes Algorithm:

Theorem 5 [6] *Let* $\bar{A} \in R^{m,n}$ *with* $p = \max\{m,n\}$ *and* $R = \mathbb{Z}$ *and* $R = \mathbb{F}[x]$*, respectively.*
a) *In the parallel Kannan Bachem Algorithm* $O(p^4 \log_2(p\|\bar{A}\|_\infty))$ *and* $O(p^4\lceil\bar{A}\rceil_{\deg})$ *BROADCAST operations are performed, respectively, and* $O(p^5 \log_2(p\|\bar{A}\|_\infty))$ *and* $O(p^5\lceil\bar{A}\rceil_{\deg})$ *ring elements are sent with BROADCAST, respectively. Further* $O(q^2p^2 \log_2(p\|\bar{A}\|_\infty))$ *and* $O(q^2p^2\lceil\bar{A}\rceil_{\deg})$ *SEND operations are performed, respectively, and* $O(p^4 \log_2(p\|\bar{A}\|_\infty))$ *and* $O(p^4\lceil\bar{A}\rceil_{\deg})$ *ring elements are sent with SEND, respectively.*
b) *In the parallel Hartley Hawkes Algorithm* $O(p^2 \log_2(p\|\bar{A}\|_\infty))$ *and* $O(p^2\lceil\bar{A}\rceil_{\deg})$ *BROADCAST operations are performed, respectively, and* $O(p^3 \log_2(p\|\bar{A}\|_\infty))$ *and* $O(p^3\lceil\bar{A}\rceil_{\deg})$ *ring elements are sent with BROADCAST, respectively. Further* $O(q^2p^2 \log_2(p\|\bar{A}\|_\infty))$ *and* $O(q^2p^2\lceil\bar{A}\rceil_{\deg})$ *SEND operations are performed, respectively, and* $O(p^4 \log_2(p\|\bar{A}\|_\infty))$ *and* $O(p^4\lceil\bar{A}\rceil_{\deg})$ *ring elements are sent with SEND, respectively.*

We obtain that the Triangular Band Matrix Algorithm gives a considerably better complexity than both algorithms.

5 Experiments with the parallel Triangular Band Matrix Algorithm

5.1 Implementation

The algorithms of this paper were implemented in the language C++ with the compiler *ibmcxx*, version 3.6.4.0. The experiments were made under AIX 4.3.3 on a POWER3-II 375 MHz processor of a IBM RS/6000 SP-SMP computer with main memory 1024 MB. We have implemented the parallel programs with *mpich*, version 1.1.1, an implementation of the message passing library MPI. It is a distributed memory computer which consists of altogether 128 POWER3-II multi processors with 1024 MB main memory for each processor. We use up to 64 processors and 2 processors are on a node. Every process of our parallel program runs on one of these processors.
The execution times are presented as "hours:minutes: seconds". If for some matrices a main part of the main memory is needed, we additionally give the used memory in MB. As we will obtain, in the Kannan Bachem Algorithm and in the Triangular Band Matrix Algorithm the main part of the execution time is needed for the first HNF algorithm. So these algorithms behave similarly and we do not need to compare the Triangular Band Matrix Algorithm with the Hartley Hawkes Algorithm, as the relations are the same like between the Kannan Bachem Algorithm and the Hartley Hawkes Algorithm (the Kannan Bachem Algorithm gives better performance for the ring $\mathbb{Z}$ and the Hartley Hawkes Algorithm gives better performance for the ring $= \mathbb{F}_2[x]$, see [6]). To compare the execution times

for the Kannan Bachem Algorithm [6] and for the Triangular Band Matrix Algorithm 3, we give only the execution time after the first HNF algorithm.

5.2 Efficiency

An important criterion for the quality of a parallel algorithm is the *efficiency* E, dependent on the number of processes q, which we define as $E(q) = \frac{T_s}{q \cdot T_q}$, where T_q is the execution time for the parallel algorithm with q processes and T_s the execution time for the corresponding original algorithm. The better is the parallel program the larger is the efficiency. The efficiency is at most 1.
We compute the Smith normal form of a matrix of middle size with the corresponding transformation matrices on 1, 2, 4, 8, 16, 32 and 64 processes with the parallel Kannan Bachem Algorithm and the parallel Triangular Band Matrix Algorithm 3, and we also use the corresponding original algorithms. We give the execution time for the second parts only. We give the efficiency of the whole algorithm in % and in brackets the efficiency of the second parts.

Matrices over the ring $\mathbb{Z}$

For the experiments we use the matrices $A_n = (a_{s,t} = (s-1)^{t-1} \mod n)$ for $1 \leq s,t \leq n$. These matrices have full rank, if n is a prime. The results are shown in Table 1. If we compare the second parts of the Kannan Bachem Algorithm and the Triangular Band Matrix Algorithm, we obtain, that the execution time for the Triangular Band Matrix Algorithm is half of the execution time for the Kannan Bachem Algorithm. This results in a little larger overall efficiency of Algorithm 3. The second parts of both algorithms are roughly constant, i.e. not very efficient. But you could not expect this, as the Kannan Bachem Algorithm works in the HNF and LHNF computations after the first HNF computation with matrices with small entries and many zeros. So the communication outweighs the "easy" calculation operations. The auxiliary algorithm PAR-DIAGTOSMITH like other steps of the Triangular Band Matrix Algorithm is parallely performed only for the computation of the transformation matrices, i.e. sometimes only one process acts on the matrix A, whereas the other ones sleep.

Matrices over the ring $\mathbb{F}_2[x]$

For the experiments we use the matrices $B_n^q = (b_{s,t} = p_{s-1}^{t-1} \mod q)$ for $1 \leq s,t \leq n$ with $(p_s(x))_{s \geq 0} = (0, 1, x, x+1, x^2, x^2+1, \cdots)$ and $q(x) = x^{10} + x^9 + x^8 + x^5 + 1$ (irreducible). For most $q(x)$ and for the special $q(x)$ we consider, these matrices have full rank. In Table 2 we receive a similar result like for the ring $\mathbb{Z}$. The efficiency of the second part of the Triangular Band Matrix Algorithm is quite better than that of the Kannan Bachem Algorithm.

Matrix	Proc.	HNF Alg.	KB Alg.	TBM Alg.
A_{389}	1 (orig.)	05:42:13	00:01:24	00:00:42
A_{389}	1 (par.)	05:46:46	00:03:26 98 (41) %	00:01:43 98 (41) %
A_{389}	2	02:54:28	00:02:33 97 (27) %	00:01:15 98 (28) %
A_{389}	4	01:29:07	00:02:14 94 (16) %	00:01:00 95 (18) %
A_{389}	8	00:47:15	00:02:11 87 (8) %	00:00:56 89 (9) %
A_{389}	16	00:25:12	00:02:10 78 (4) %	00:01:02 82 (4) %
A_{389}	32	00:14:48	00:02:17 63 (2) %	00:01:10 67 (2) %
A_{389}	64	00:09:34	00:02:31 44 (1) %	00:01:22 49 (1) %

Table 1. Efficiency of the Kannan Bachem Algorithm and the Triangular Band Matrix Algorithm for the ring $\mathbb{Z}$

Matrix	Proc.	HNF Alg.	KB Alg.	TBM Alg.
B_{400}^q	1 (orig.)	10:30:32	00:00:25	00:00:32
B_{400}^q	1 (par.)	10:49:17	00:02:22 97 (18) %	00:01:03 97 (51) %
B_{400}^q	2	05:35:44	00:01:18 94 (16) %	00:00:36 94 (44) %
B_{400}^q	4	02:54:15	00:01:06 90 (9) %	00:00:28 90 (29) %
B_{400}^q	8	01:30:00	00:01:08 87 (5) %	00:00:29 87 (14) %
B_{400}^q	16	00:47:20	00:01:18 81 (2) %	00:00:36 82 (6) %
B_{400}^q	32	00:26:26	00:01:22 71 (1) %	00:00:38 73 (3) %
B_{400}^q	64	00:16:23	00:01:38 55 (0) %	00:00:50 57 (1) %

Table 2. Efficiency of the Kannan Bachem Algorithm and the Triangular Band Matrix Algorithm for the ring $\mathbb{F}_2[x]$

5.3 Large example matrices over the ring $\mathbb{Z}$

Table 3 shows that the Triangular Band Matrix Algorithm is able to compute the Smith normal form of the same large matrices like for the Kannan Bachem Algorithm. Comparing the second parts, in most cases the Triangular Band Matrix Algorithm 3 needs less than half of the execution time of the Kannan Bachem Algorithm.

Matrix	Proc.	HNF Alg.	KB Alg.	TBM Alg.
A_{967}	64	09:07:00	00:37:41	00:06:32
A_{983}	64	10:31:53	00:36:16	00:02:09
A_{997}	64	10:21:51	00:39:56	00:20:08
A_{1013}	64	11:12:28	00:39:41	00:17:04
A_{1117}	64	17:16:10	00:59:04	00:33:52
A_{1223}	64	29:37:55	01:06:39 129 MB	00:07:34 129 MB

Table 3. Kannan Bachem Algorithm and Triangular Band Matrix Algorithm with 64 processes for the ring $\mathbb{Z}$

6 Conclusion

We have developed a new Smith normal form algorithm both for the rings $\mathbb{Z}$ and $\mathbb{F}[x]$, which is also able to compute the corresponding transformation matrices. Parallelizing this algorithm, we obtain theoretical and practical improvements, in comparison to previous algorithms.

References

[1] M. Giesbrecht, Fast Computation of the Smith Normal Form of an Integer Matrix, *Proceedings of the 1995 international symposium on symbolic and algebraic computation , ISSAC '95*, Montreal, Canada, July 10-12, 1995, 110-118, ACM Press, New York.

[2] J.L. Hafner and K.S. McCurley, Asymptotically Fast Triangularization of Matrices over Rings, *SIAM J. Computing* **20**(6) (1991) 1068-1083.

[3] B. Hartley and T.O. Hawkes, *Rings, Modules and Linear Algebra* (Chapman and Hall, London, 1976).

[4] G. Havas and L.S. Sterling, Integer Matrices and Abelian Groups, *Lecture Notes in Computer Science* **72** (1979) 431-451, Springer-Verlag, New York.

[5] C. Hermite, Sur l'introduction des variables continues dans la théorie des nombres, *J. Reine Angew. Math.* **41** (1851) 191-216.

[6] G. Jäger, Parallel Algorithms for Computing the Smith Normal Form of Large Matrices, *Lecture Notes in Computer Science* **2840** (2003) 170-179, Springer-Verlag, New York.

[7] E. Kaltofen, M. S. Krishnamoorthy and B. D. Saunders, Fast parallel computation of Hermite and Smith forms of polynomial matrices, *SIAM J. Algebraic and Discrete Methods* **8** (1987) 683-690.

[8] E. Kaltofen, M. S. Krishnamoorthy and B. D. Saunders, Parallel Algorithms for Matrix Normal Forms, *Linear Algebra and its Applications* **136** (1990) 189-208.

[9] R. Kannan and A. Bachem, Polynomial Algorithms for Computing the Smith and Hermite Normal Forms of an Integer Matrix, *SIAM J. Computing* **8**(4) (1979) 499-507.

[10] H.-J. Kowalsky and G.O. Michler, *Lineare Algebra* (de Gruyter, Berlin, 1998).

[11] I.S. Pace and S. Barnett, Efficient algorithms for linear system calculations , Part I, Smith form and common divisor of polynomial matrices, *Int. J. Systems Sci.* **5** (1974) 403-411.

[12] C.C. Sims, *Computation with finitely presented groups* (Cambridge University Press, 1994).

[13] H.J.S. Smith, On Systems of Linear Indeterminate Equations and Congruences, *Philos. Trans. Royal Soc. London* **151** (1861) 293-326.

[14] A. Storjohann, Algorithms for Matrix Canonical Forms. Ph. D. Thesis, ETH Zürich, 2000.

[15] A. Storjohann, Computing Hermite and Smith normal forms of triangular integer matrices, *Linear Algebra and its Applications* **282** (1998) 25-45.

[16] A. Storjohann, Near Optimal Algorithms for Computing Smith Normal Forms of Integer Matrices, *Proceedings of the 1996 international symposium on symbolic and algebraic computation, ISSAC '96*, Zürich, Switzerland, July 24-26, 1996, 267-274, ACM Press, New York.

[17] A. Storjohann and G. Labahn, A Fast Las Vegas Algorithm for Computing the Smith Normal Form of a Polynomial Matrix, *Linear Algebra and its Applications* **253** (1997) 155-173.

[18] G. Villard: Fast parallel computation of the Smith normal form of poylynomial matrices, *Proceedings of the 1994 international symposium on symbolic and algebraic computation, ISSAC '94*, Oxford, UK, July 20-22, 1994, 312-317, ACM Press, New York.

[19] C. Wagner, Normalformenberechnung von Matrizen über euklidischen Ringen, Ph. D. Thesis, Institut für Experimentelle Mathematik, Universität/GH Essen, 1997.