Parallel Algorithms for Computing the Smith Normal Form of Large Matrices

Gerold Jäger

Mathematisches Seminar der Christian-Albrechts-Universität zu Kiel,
Christian-Albrechts-Platz 4, D-24118 Kiel, Germany
gej@numerik.uni-kiel.de

Extended Abstract

Abstract. Smith normal form computation has many applications in group theory, module theory and number theory. As the entries of the matrix and of its corresponding transformation matrices can explode during the computation, it is a very difficult problem to compute the Smith normal form of large dense matrices. The computation has two main problems: the high execution time and the memory requirements, which might exceed the memory of one processor. To avoid these problems, we develop two parallel Smith normal form algorithms using MPI. These are the first algorithms computing the Smith normal form with corresponding transformation matrices, both over the rings $\mathbb{Z}$ and $\mathbb{F}[x]$. We show that our parallel algorithms both have a good efficiency, i.e. by doubling the processes, the execution time is nearly halved, and succeed in computing the Smith normal form of dense example matrices over the rings $\mathbb{Z}$ and $\mathbb{F}_2[x]$ with more than thousand rows and columns.

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 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.

The Smith normal form plays an important role in the theory of finite Abelian groups and in the theory of finitely generated modules over principal ideal rings. For many applications, for example the integer solutions of an integer system of linear equations, the transformation matrices describing the unimodular operations are important as well.

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 ([2] for $R = \mathbb{Z}$, [16] for $R = \mathbb{Q}[x]$). Deterministic algorithms for $R = \mathbb{Z}$ often use modular techniques ([3], [5], [12, Chapter 8.4], [14], [15]). Unfortunately, these algorithms are unable to compute the corresponding transformation matrices.

We shortly introduce the two most important Smith normal form algorithms which work for both the ring $\mathbb{Z}$ and the ring $\mathbb{F}[x]$ and which are able to compute the corresponding transformation matrices. Both the algorithm of Kannan, Bachem [9] and the algorithm of Hartley, Hawkes [4] compute the diagonal form first and finally with an elementary algorithm the Smith normal form.

The disadvantage of these algorithms is that during the computation the entries of the matrix and the corresponding transformation matrices can be very large, even exponential [1]. For high-dimensional matrices with large entries this leads to high execution times and memory problems which can be solved by parallelization.

In [7], [8] and [17] parallel probabilistic algorithms are introduced for the ring $\mathbb{F}[x]$, but without experimental results, and in [11] a parallel algorithm is described, which only works for characteristic matrices.

In this paper we parallelize the general algorithms of Kannan, Bachem and Hartley, Hawkes for rectangular matrices. The main problem is how to uniformly distribute a large matrix to many processes. The key observation is that a row distribution is more efficient, if we use column operations, and column distribution is more efficient, if we use row operations. As we use both row and column operations, we develop an auxiliary algorithm which transforms a row distributed matrix into a column distributed one and vice versa. We estimate the parallel operations of our two algorithms and see that the complexity of the parallel Hartley-Hawkes-Algorithm is better than that of the parallel Kannan-Bachem-Algorithm.

We implement the algorithms and test it for large dense matrices over the rings $\mathbb{Z}$ and $\mathbb{F}_2[x]$. The experiments show that the parallel Kannan-Bachem-Algorithm leads to better results for the ring $\mathbb{Z}$ and the parallel Hartley-Hawkes-Algorithm to better results for the ring $\mathbb{F}_2[x]$. Considering medium-sized matrices, we see that the algorithms have a good efficiency, even for 64 processes. The algorithms are also able to compute the Smith normal form with its corresponding transformation matrices of very large matrices in reasonable time. Because of the memory requirements, the program package MAGMA is not able to do such computations.

2 Preliminaries

2.1 Notations and Definitions

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:
a) The set $\mathbb{Z}$ of integers
We choose $\phi := |\cdot|$, $\mathcal{R} := \mathbb{N}_0$. For $a \in \mathbb{R}$ let $\lfloor a \rfloor$ be the largest integer $\leq a$. With the above notations we define $\psi(a, b) := r = a - \lfloor a/b \rfloor \cdot b$. For $A \in \mathbb{Z}^{m,n}$ let $\|A\|_\infty := \max_{1 \leq i,j \leq n}\{|A_{i,j}|\}$.
b) The polynomial ring $\mathbb{F}[x]$ with a field $\mathbb{F}$

We choose $\phi := \deg$, $\mathcal{R} := \{\text{Monic polynomials over } \mathbb{F}[x]\}$. With the above notations we define $\psi(a,b) := r$, where r is uniquely determined by polynomial division of a and b. For $A \in \mathbb{F}[x]^{m,n}$ let $\lceil A \rceil_{\deg} := \max_{1 \le i,j \le n} \{\deg(A_{i,j})\}$.

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 matrix $A \in R^{m,n}$ with rank r is in Hermite normal form (HNF), if the following conditions hold:*

a) $\exists i_1, \cdots, i_r$ *with* $1 \le i_1 < \cdots < i_r \le m$ *with* $A_{i_j,j} \in \mathcal{R} \setminus 0$ *for* $1 \le j \le r$.
b) $A_{i,j} = 0$ *for* $1 \le i \le i_j - 1,\ 1 \le j \le r$.
c) *The columns* $r+1, \cdots, n$ *are zero.*
d) $A_{i_j,l} = \psi(A_{i_j,l}, A_{i_j,j})$ *for* $1 \le l < j \le r$.

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

Theorem 1. [6] *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 matrix $A \in R^{m,n}$ with rank r is in Smith normal form (SNF), if the following conditions hold:*

a) *A is a diagonal matrix.*
b) $A_{i,i} \in \mathcal{R} \setminus \{0\}$ *for* $1 \le i \le r$.
c) $A_{i,i} \mid A_{i+1,i+1}$ *for* $1 \le i \le r-1$.
d) $A_{i,i} = 0$ *for* $r+1 \le i \le \min\{m,n\}$.

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. The matrices U, V are called the corresponding left hand and right hand transformation matrix for the Smith normal form.*

In the following Smith normal form algorithms we receive the corresponding transformation matrices, if we apply the row operations to E_m and the column operations to E_n.

2.2 Algorithm DIAGTOSMITH

In the SNF algorithms we use an elementary algorithm DIAGTOSMITH which computes the Smith normal form of a matrix in diagonal form [10, p. 318, Hilfssatz 11.5.14]. Two neighboring diagonal elements are substituted by its gcd and its lcm, until the matrix is in Smith normal form. The algorithm needs $\min\{m,n\}^2$ gcd computations.

2.3 Kannan-Bachem-Algorithm

The algorithm of Kannan and Bachem ([9]) alternately computes the Hermite normal form and the left Hermite normal form, until the matrix is in diagonal form. At the end the algorithm DIAGTOSMITH is applied.

Remark 1. Let $A \in R^{m,n}$ with $p = \max\{m,n\}$. Then the HNF and LHNF procedure of the Kannan-Bachem-Algorithm is executed $O(p^2 \log_2(p\|A\|_\infty))$ times for $R = \mathbb{Z}$ and $O(p^2\lceil A\rceil_{\deg})$ times for $R = \mathbb{F}[x]$.

2.4 Hartley-Hawkes-Algorithm

For i,j with $1 \le i \le m,\ 1 \le j \le n$ ROWGCD (A,i,j) transforms A so that $A_{i,j}^{\text{new}} = \gcd(A_{i,j}^{\text{old}}, A_{i,j+1}^{\text{old}}, \cdots, A_{i,n}^{\text{old}}) \in \mathcal{R}$, $A_{i,j+1}^{\text{new}} = \cdots = A_{i,n}^{\text{new}} = 0$. This is done by subtracting the multiple of a column from a different column very often. The procedure COLGCD is analogously defined with the role of rows and columns exchanged. The algorithm of Hartley and Hawkes [4, p. 112] alternately uses the procedures ROWGCD (l,l) and COLGCD (l,l), until the first $l-1$ rows and columns have diagonal form. Finally the algorithm DIAGTOSMITH is used again.

Remark 2. Let $A \in R^{m,n}$ with $p = \max\{m,n\}$. Then the ROWGCD and COLGCD procedure of the Hartley-Hawkes-Algorithm is executed at most $O(p^2 \log_2(p\|A\|_\infty))$ times for $R = \mathbb{Z}$ and at most $O(p^2\lceil A\rceil_{\deg})$ times for $R = \mathbb{F}[x]$.

3 Parallelization of the Smith normal form algorithms

3.1 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 **BROADCAST-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. It is straightforward to put different rows or different columns of a matrix to one process. For algorithms, in which mainly column operations are used, a column distribution is not sensible, as for column additions with multiplicity the columns involved in computations mostly belong to different processes, so that for each such computation a whole column has to be sent. So we decide to distribute rows, if column operations are used, and columns, if row operations are used. As for Smith normal form algorithms both operations are used, we have to switch between both distributions (see the procedures PAR-ROWTOCOL and PAR-COLTOROW).
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 z has as input a matrix $A \in R^{k(z),n}$ with $\sum_{z=1}^{q} k(z) = m$. 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 uniform 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. Analogously, we choose a column distribution for the left hand transformation matrix U.

Theorem 3. [18] *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.*

3.2 Auxiliary algorithm PAR-DIAGTOSMITH

As the original procedure DIAGTOSMITH is very fast in practice, it is sufficient to parallelize it 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 use two versions of this algorithm, one for row distribution and one for column distribution.

3.3 Auxiliary algorithms PAR-ROWTOCOL and PAR-COLTOROW

The algorithm PAR-ROWTOCOL 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. For a process x the SEND operation has the following form:

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

For a process x the SEND-RECEIVE operation has the following form:

$$\text{SEND-RECEIVE}\,\{B_{s,t} \mid 1 \leq s \leq m,\, 1 \leq t \leq k'(x)\}\ \ \text{FROM}\ \ \text{ROW-TASK}\,(s)$$

This algorithm does not need to be applied to the corresponding 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. We obtain the algorithm PAR-COLTOROW from the algorithm PAR-ROWTOCOL by exchanging the role of rows and columns.

3.4 Parallel Kannan-Bachem-Algorithm

Algorithm 1
INPUT Number of processes q, number of its own process z,
$A \in R^{k(z),n}$, $\sum_{z=1}^{q} k(z) = m$ and whole matrix $\bar{A} \in R^{m,n}$
1 *WHILE ($\bar{A}$ is not in diagonal form)*
2 $A = PAR\text{-}HNF(A)$
3 $B = PAR\text{-}ROWTOCOL(A)$ $\left(B \in R^{m,k'(z)}, \sum_{z=1}^{q} k'(z) = n\right)$
4 $B = PAR\text{-}LHNF(B)$
5 $A = PAR\text{-}COLTOROW(B)$
6 $A = PAR\text{-}DIAGTOSMITH(A)$
OUTPUT $A = PAR\text{-}SNF(A) \in R^{k(z),n}$

Theorem 4. *Let $\bar{A} \in R^{m,n}$ with $p = \max\{m, n\}$ and $R = \mathbb{Z}$ and $R = \mathbb{F}[x]$, respectively. In Algorithm 1 $O(p^4 \log_2(p\|\bar{A}\|_\infty))$ 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 $O(q^2 p^2 \log_2(p \|\bar{A}\|_\infty))$ and $O(q^2 p^2 \lceil \bar{A} \rceil_{\deg})$ 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.*

Proof: Follows from Remark 1 and Theorem 3 and the definitions of PAR-ROWTOCOL and PAR-COLTOROW. □

3.5 Parallel Hartley-Hawkes-Algorithm

Algorithm 2
INPUT Number of processes q, number of its own process z,
$A = [a_1, \cdots, a_{k(z)}]^T \in R^{k(z),n}$ and $\sum_{z=1}^{q} k(z) = m$
1 $l = 1$
2 *WHILE* $l \leq \min\{m, n\}$
3 $y = ROW\text{-}TASK(l)$
4 $h = ROW\text{-}NUM(l)$
5 *IF* $y = z$
6 *THEN BROADCAST* a_h
7 *ELSE BROADCAST-RECEIVE* v *FROM* y
8 *Insert* v *as* h*-th row vector of* A
9 *IF NOT* $(A_{h,j})_{l+1 \leq j \leq n} = 0$

10 *THEN* $A = ROWGCD(A, h, l)$
11 *IF NOT* $y = z$
12 *THEN Remove* v *as* h*-th row vector of* A
13 $B = PAR\text{-}ROWTOCOL(A)$ $\left(B \in R^{m,k'(z)}, \sum_{z=1}^{q} k'(z) = n\right)$
14 $y = COL\text{-}TASK(l)$
15 $h = COL\text{-}NUM(l)$
16 *IF* $y = z$
17 *THEN BROADCAST* b_h *(the* h*-th column of* B*)*
18 *ELSE BROADCAST-RECEIVE* v *FROM* y
19 *Insert* v *as* h*-th column vector of* B
20 *IF* $(B_{i,h})_{l+1 \le i \le m} = 0$
21 *THEN* $l = l + 1$
22 *ELSE* $B = COLGCD(B, l, h)$
23 *IF NOT* $y = z$
24 *THEN Remove* v *as* h*-th column vector of* B
25 $A = PAR\text{-}COLTOROW(B)$
26 $A = PAR\text{-}DIAGTOSMITH(A)$
OUTPUT $A = PAR\text{-}SNF(A) \in R^{k(z),n}$

Correctness: The original algorithm consists of the steps 1, 2, 9, 10, 20, 21, 22 and 26 with l instead of h and A instead of B. We do the same steps as in the original algorithm, but on the processes owning the current elements.
At the beginning the matrix is row distributed. As in stage l of the WHILE-loop the original l-th row is decisive for the row operations which have to be done, in the steps 3 to 7 the l-th row is sent from the process it lies on to all other processes. In step 8 this row is inserted into the h-th place behind all rows, which are in the whole matrix above the l-th row. In step 9 and 10 the row at the h-th place and the rows behind it are transformed on all processes corresponding to the original algorithm. In the steps 11 and 12 the row is removed from all processes which have received it. In step 13 the matrix is transformed from row distribution to column distribution. In the steps 14 to 25 we do the analogous steps for the column distributed matrix. □

Theorem 5. *Let* $\bar{A} \in R^{m,n}$ *with* $p = \max\{m, n\}$ *and* $R = \mathbb{Z}$ *and* $R = \mathbb{F}[x]$*, respectively. In Algorithm 2* $O(p^2 \log_2(p\|\bar{A}\|_\infty))$ *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* $O(q^2 p^2 \log_2(p \ \|\bar{A}\|_\infty))$ *and* $O(q^2 p^2 \lceil \bar{A} \rceil_{\deg})$ *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.*

Proof: Follows from Remark 2 and the definitions of PAR-ROWTOCOL and PAR-COLTOROW. □

In comparison to Algorithm 1 the complexity of the BROADCAST operations improves by a factor of p^2, whereas the complexity of the SEND operations stays unchanged.

4 Experiments with the parallel versions of the Smith normal form algorithms

The original algorithms of this paper were implemented in the language C++ with the compiler *ibmcxx*, version 3.6.4.0 and the parallel programs with *mpich*, version 1.1.1, an implementation of the message passing library MPI. The experiments were made under AIX 4.3.3 on a POWER3-II 375 MHz processor of a IBM RS/6000 SP-SMP computer at the SSC Karlsruhe. 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, where 2 processors belong to a node. Every process of our parallel program runs on one of these processors. Additionally, we compare our results with the program package MAGMA V2.5-1, started under AIX 4.2 on a POWER2 66 MHz computer with main memory 1024 MB.

4.1 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 with corresponding transformation matrices of a matrix of medium size on 1, 2, 4, 8, 16, 32 and 64 processes with the parallel Algorithms 1 and 2, and we also use the corresponding original algorithms. As parallel HNF algorithm we have implemented the algorithm of Theorem 3. For every parallel SNF computation we give the efficiency in %.

Matrices over the ring $\mathbb{Z}$: For the experiments we use the following class of 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, and have rather large entries.

Pro.	1 (orig.)	1 (par.)	2	4	8	16	32	64	MAG.
Ti.	05:43:37	05:50:12	02:57:01	01:31:21	00:49:26	00:27:22	00:17:05	00:12:05	38:50:47
Eff.	-	98 %	97 %	94 %	87 %	78 %	63 %	44 %	-

Table 1. Execution time and efficiency of the parallel Kannan-Bachem-Algorithm, applied to the matrix A_{389}

The Hartley-Hawkes-Algorithm 2 produces a memory overflow even with 64 processes. The efficiency of the Kannan-Bachem-Algorithm 1 is rather high for up to 16 processes and lower for 32 and 64 processes.

Matrices over the ring $\mathbb{F}_2[x]$: We use the following class of 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 the irreducible polynom $q(x) = x^{10} + x^9 + x^8 + x^5 + 1$. These matrices also have full rank with large entries.

Proc.	1 (orig.)	1 (par.)	2	4	8	16	32	64
Time (KB)	10:30:57	10:51:39	05:37:02	02:55:21	01:31:08	00:48:38	00:27:48	00:18:01
Eff. (KB)	-	97 %	94 %	90 %	87 %	81 %	71 %	55 %
Time (HH)	05:47:01	06:10:55	03:04:36	01:33:14	00:47:24	00:24:49	00:13:51	00:08:34
Eff. (HH)	-	94 %	94 %	93 %	92 %	87 %	78 %	63 %

Table 2. Execution time and efficiency of the parallel Kannan-Bachem-Algorithm and Hartley-Hawkes-Algorithm, applied to the matrix B^q_{400}

We see, that the Hartley-Hawkes-Algorithm 2 is the better SNF algorithm. The efficiency is from 16 processes on higher than for the ring $\mathbb{Z}$. MAGMA was not able to compute the Smith normal form of B^q_{400} with corresponding transformation matrices.

4.2 Large example matrices

For the rings $\mathbb{Z}$ and $\mathbb{F}_2[x]$ we want to find out the maximum number of rows and columns of an input matrix we are able to compute the Smith normal form for. For some examples we additionally list the used memory in MB.

Matrices over the ring $\mathbb{Z}$: For the ring $\mathbb{Z}$ the Kannan-Bachem-Algorithm 1 is the best parallel algorithm (compare Table 1).

Mat.	A_{967}	A_{983}	A_{997}	A_{1013}	A_{1117}	A_{1223}
Time	09:44:41	11:08:09	11:02:47	11:52:09	18:15:14	30:44:31 (129 MB)

Table 3. Execution time and efficiency of the parallel Kannan-Bachem-Algorithm with 64 processes

We succeed in computing the Smith normal form of a (1223×1223) matrix with maximum absolute value 1222. The available memory of 1024 MB is not fully used, i.e. if we have more time, it is possible to compute the Smith normal form of larger matrices.

Matrices over the ring $\mathbb{F}_2[x]$: For the ring $\mathbb{F}_2[x]$ the Hartley-Hawkes-Algorithm 2 is the best parallel algorithm (compare Table 2). Additionally we use the test matrices $D^{(1)}_{1792}, D^{(2)}_{1792}, D^{(3)}_{1792}, D^{(4)}_{1792}$. These are characteristic matrices with full rank and elementary divisors which are mostly no unit ([11, example 7.3]).

Mat.	$D^{(1)}_{1792}$	$D^{(2)}_{1792}$	$D^{(3)}_{1792}$	$D^{(4)}_{1792}$	B^q_{1024}
Time	01:25:50	01:04:38	01:31:30	01:25:41 (82 MB)	07:04:31 (170 MB)

Table 4. Execution time and efficiency of the parallel Hartley-Hawkes-Algorithm with 64 processes

We succeed in computing the Smith normal form of a (1024×1024) matrix and of the four (1792×1792) matrices. The matrix B^q_{1024} is more difficult, as it contains polynomials up to degree 9, whereas the matrices $D^{(i)}_{1792}$ only contain polynomials of degree 0 and 1. The matrix B^q_{1024} needs more memory than all other large matrices considered. MAGMA was not able to compute the Smith normal form with corresponding transformation matrices of one of these large matrices.

References

1. X.G. Fang, G. Havas: On the worst-case complexity of integer gaussian elimination, Proc. of ISSAC (1997) 28-31.
2. M. Giesbrecht: Fast Computation of the Smith Normal Form of an Integer Matrix, Proc. of ISSAC (1995) 110-118.
3. J.L. Hafner, K.S. McCurley: Asymptotically Fast Triangularization of Matrices over Rings, SIAM J. Computing **20**(6) (1991) 1068-1083.
4. B. Hartley, T.O. Hawkes: Rings, Modules and Linear Algebra, Chapman and Hall, London (1976).
5. G. Havas, L.S. Sterling: Integer Matrices and Abelian Groups, Lecture Notes in Computer Science **72**, Springer-Verlag, New York (1979) 431-451.
6. C. Hermite: Sur l'introduction des variables continues dans la théorie des nombres, J. Reine Angew. Math. **41** (1851) 191-216.
7. E. Kaltofen, M.S. Krishnamoorthy, 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, B.D. Saunders: Parallel Algorithms for Matrix Normal Forms, Linear Algebra and its Applications **136** (1990) 189-208.
9. R. Kannan, 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, G.O. Michler: Lineare Algebra, de Gruyter, Berlin (1998).
11. G.O. Michler, R. Staszewski: Diagonalizing Characteristic Matrices on Parallel Machines, Preprint 27, Institut für Experimentelle Mathematik, Universität/GH Essen (1995).
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: Computing Hermite and Smith normal forms of triangular integer matrices, Linear Algebra and its Applications **282** (1998) 25-45.
15. A. Storjohann: Near Optimal Algorithms for Computing Smith Normal Forms of Integer Matrices, Proc. of ISSAC (1996) 267-274.
16. A. Storjohann, 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.
17. G. Villard: Fast parallel computation of the Smith normal form of poylynomial matrices, Proc. of ISSAC (1994) 312-317.
18. C. Wagner: Normalformenberechnung von Matrizen über euklidischen Ringen, Ph.D. Thesis, Institut für Experimentelle Mathematik, Universität/GH Essen (1997).