INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Parallelism in Hermite and Smith normal forms

Tomáš Hrúz et Dominique Fortin

N° 2077

Octobre 1993

PROGRAMME 1

Architectures parallèles,

bases de données,

réseaux et systèmes distribués

Rapport de recherche

1994

ISSN 0249-6399

Parallelism in Hermite and Smith normal forms

Tomáš Hrúz * et Dominique Fortin**

Programme 1 — Architectures parallèles, bases de données, réseaux et systèmes distribués
Projet Archi

Rapport de recherche n°2077 — Octobre 1993 — 40 pages

Abstract: The Smith and Hermite normal forms play an important role in various fields of investigations. In many applications it is crucial to compute the Smith or Hermite normal form of an integral matrix. A construction of a special sequence of integral matrices based on the twin primes is presented. We suggest that this construction could show a method how to obtain a lower bound for triangular unimodular matrices in the terms of elementary operations. We also define a class of parallel networks for the computation of a gcd of n numbers. An analogy of the zero-one principle for comparison networks is derived for the gcd networks.

Key-words: Smith normal form, Hermite normal form, integer matrices, computational complexity, Greatest Common Divisor, parallel algorithms

(Résumé : tsvp)

*SLOVAK TECHNICAL UNIVERSITY, Faculty of Mechanical Engineering, Department of Automatic Control and Measurement Námestie Slobody 17, 812 31 Bratislava, Slovak Republic, Phone: (42 7) 497193, e-mail: hruz@cvt.stuba.sk
**Dominique.Fortin@inria.fr

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
Téléphone : (33 1) 39 63 55 11 – Télécopie : (33 1) 39 63 53 30

Du parallélisme dans le calcul des formes normales de Hermite et Smith

Résumé : Les formes normales de Smith et Hermite jouent un rôle important dans divers domaines de recherche. Dans de nombreuses applications, il est crucial de calculer la forme normale de Smith ou d'Hermite d'une matrice d'entiers. La construction d'une séquence particulière de matrices d'entiers basée sur des couples de nombres premiers voisins est présentée. Nous suggérons comment cette construction peut faire apparaître une méthode pour obtenir une borne inférieure, en terme d'opérations élémentaires, dans le cas de matrices unimodulaires triangulaires. Nous définissons aussi une classe de réseaux parallèles pour le calcul du gcd de n nombres. Un principe zéro-un, analogue à celui utilisé dans les réseaux de comparaisons, est décrite pour les réseaux gcd.

Mots-clé : Forme normale de Smith , Forme normale de Hermite, matrices entières, complexité , Plus Grand Commun Diviseur, algorithmes parallèles

Contents

1 Introduction

From the point when we have started an investigation of the parallelism in Hermite normal form and Smith normal form computation several branches of interest have emerged and the arrangement of this report is concordant with them. As is usual in this field we have finally drawn our attention to the gcd computation of n numbers.

The first branch deals with the lower bound for the number of elementary operations in Hermite normal form computation. We have decomposed the problem into two parts: the investigation of a lower bound for a triangular unimodular matrix and a general unimodular matrix investigation (a combination with a gcd lower bound). Now we are in the process of finishing the first part. This branch in covered in the section "Upper triangular unimodular matrices".

The second branch concerns the parallel structures for gcd computation of n integers. We started with well known structures such as permutation and sorting networks to introduce a structure we call a gcd network. The computer experiments and our current understanding of the problem have convinced us that this kind of structures could provide efficient algorithms for the gcd computation. As has been mentioned several times the gcd problem for integers is different from the same task on polynomials over various kinds of fields with respect to parallelism. R. Kannan noted this by saying: "Integers seem to be inherently sequential". In our investigations this problem has emerged in a form of a rapid degeneration of the integer sample causing a situation where only a very small part of the network does worthwhile work. The goal here is to design networks capable of efficiently dealing with degenerated samples and to derive some estimations of a worst case. The experimental work is also not completely finished. We wish to have more balanced experiments with respect to the granularity of computation.

The third branch is purely algebraic and is at the very beginning. In fact it concerns the structure of a general unimodular group from the point of view which is interesting in our searching of algorithmic solutions.

The first author would like to thank the ARCHI team and especially Mr. D. Tusera for ongoing support for this exciting investigation and for the supportive environment they have provided.

1.1 Hermite normal form, Smith normal form

An integral matrix $\mathbf{U}$ with $|\det(\mathbf{U})| = 1$ is called *unimodular*. For convenience, throughout we assume matrices are $n \times n$ and nonsingular; there is a natural generalization for arbitrary integral matrices. Every nonsingular integral matrix has a lower triangular integral canonic form called Hermite normal form (see [1], theorem 2 below). The Hermite normal form of matrix $\mathbf{A}$ is a unique representation of an equivalent class of matrices with respect to congruence produced by multiplication with right unimodular matrix: $\mathbf{A} \equiv_H \mathbf{B}$ iff $\mathbf{AR} = \mathbf{B}$ where $\mathbf{R}$ is unimodular.

Another very important congruence is: $\mathbf{A} \equiv_S \mathbf{B}$ iff $\mathbf{LAR} = \mathbf{B}$ where $\mathbf{L}$,$\mathbf{R}$ are unimodular. Canonic form with respect to this congruence is called Smith normal form (see [2], theorem 3 below) and it is a diagonal integral matrix.

The Smith and Hermite normal forms play an important role in various fields of investigations [13, 10] (see also introduction to [20]). In many applications it is crucial to compute the Smith or Hermite normal form of an integral matrix.

Definition 1 (Elementary operations) *Let us have an integer matrix* $\mathbf{A}$*. We call elementary operations the following three kinds of operations on columns of the matrix* $\mathbf{A}$*:*

1. *Exchange of the columns i and j. We denote this operation by $\mathbf{S^c}(i,j)$ $i < j$ or $\mathbf{S}$ when only a type of the operation is of interest.*
2. *Multiplication of the column i by -1. This operation is denoted by $\mathbf{M^c}(i)$ or $\mathbf{M}$ when only a type of the operation is of interest.*
3. *Addition of k times $i-th$ column to the $j-th$ column. This operation is denoted by $\mathbf{A^c}(\overrightarrow{i,j},k)$ when $i < j$ and $\mathbf{A^c}(\overleftarrow{j,i},k)$ when $i > j$ or $\mathbf{A}$ when only a type of the operation is of interest.*

Theorem 2 (Hermite [1]) *Given a nonsingular $n \times n$ integral matrix $\mathbf{A}$, there exists a $n \times n$ unimodular matrix $\mathbf{R}$ such that $\mathbf{H} = \mathbf{AR}$ is a lower triangular with positive diagonal elements. Further, each off-diagonal element of $\mathbf{H}$ is nonpositive and strictly less in absolute value than the diagonal element in its row. $\mathbf{H}$ is called the Hermite normal form of $\mathbf{A}$.*

$$
\left(\begin{array}{cccccccc}
1 & & \cdot & & & & & \\
 & 1 & \cdot & & & & & \\
\cdot & \cdot & 0 & \cdot & \cdot & 1 & & \\
 & & \cdot & 1 & & \cdot & & \\
 & & \cdot & & 1 & \cdot & & \\
 & & 1 & \cdot & \cdot & 0 & & \\
 & & \cdot & & & \cdot & 1 & \\
 & & \cdot & & & \cdot & & 1 \\
\hline
 & & i & & & j & &
\end{array}\right)
\left(\begin{array}{cccccccc}
1 & & & & & & & \\
 & 1 & & & & & & \\
 & & 1 & & & & & \\
 & & & -1 & & & & \\
 & & & \cdot & 1 & & & \\
 & & & \cdot & & 1 & & \\
 & & & \cdot & & & 1 & \\
 & & & \cdot & & & & 1 \\
\hline
 & & & i & & & &
\end{array}\right)
\left(\begin{array}{cccccccc}
1 & & & & & & & \\
 & 1 & & & & & & \\
 & & 1 & \cdot & \cdot & k & & \\
 & & \cdot & 1 & & \cdot & & \\
 & & \cdot & & 1 & \cdot & & \\
 & & \cdot & & & 1 & & \\
 & & \cdot & & & \cdot & 1 & \\
 & & \cdot & & & \cdot & & 1 \\
\hline
 & & i & & \rightarrow & j & &
\end{array}\right)
$$

$$
\left(\begin{array}{cccccccc}
1 & & & & & & & \\
 & 1 & & & & & & \\
 & & 1 & & & & & \\
 & & \cdot & 1 & & & & \\
 & & \cdot & & 1 & & & \\
 & & k & \cdot & \cdot & 1 & & \\
 & & \cdot & & & \cdot & 1 & \\
 & & \cdot & & & \cdot & & 1 \\
\hline
 & & i & & \leftarrow & j & &
\end{array}\right)
$$

Figure 1:

Elementary operations in matrix form coresponding to $\mathbf{S^c}(i,j)$, $\mathbf{M^c}(i)$, $\mathbf{A^c}(\overrightarrow{i,j},k)$ resp. $\mathbf{A^c}(\overleftarrow{i,j},k)$ type operations.

It is known (see for example [25]) that the Hermite normal form of a matrix $\mathbf{A}$ is unique and also the right unimodular matrix is unique. Elementary operations from Definition 1 can be realized through multiplication of the matrix $\mathbf{A}$ by special matrices called *elementary matrices.* Elementary matrices are unimodular as can be seen in picture 1. Hermite normal form can be defined in any principal ideal ring [13], but in an Euclidean ring of integers constructive proof can be found which also shows that Hermite normal form can be constructed by elementary operations. Consequently every unimodular matrix is a product of elementary matrices. A natural generalization of an elementary operation can be defined. We call *unimodular operation* a column operation on a matrix $\mathbf{A}$ if it is equivalent to the result of right multiplying $\mathbf{A}$ by some unimodular matrix $\mathbf{R}$.

Theorem 3 (Smith [2]) *Given a nonsingular $n \times n$ integral matrix $\mathbf{A}$, there exists $n \times n$ unimodular matrices $\mathbf{L}$,$\mathbf{R}$ such that $\mathbf{S} = \mathbf{LAR}$ is a diagonal* *matrix with positive diagonal elements* $d_1, \ldots, d_n$ *such that* d_i *divides* d_{i+1} $(i = 1, \ldots, n-1)$.

The Smith normal form $\mathbf{S}$ is unique but the matrices $\mathbf{L}$,$\mathbf{R}$ are not. The *non-uniqueness* of unimodular matrices $\mathbf{L}$, $\mathbf{R}$ has important consequences as can be seen in the following sections. Matrices $\mathbf{L}$, $\mathbf{R}$ are a product of elementary matrices. The only difference with Hermite normal form is that a left multiplying a matrix with some unimodular matrix means a sequence of *row* elementary operations similar to those in Definition 1. We denote them $\mathbf{S^r}(i,j)$, $\mathbf{M^r}(i)$,$\mathbf{A^r}(\overrightarrow{i,j},k)$, $\mathbf{A^r}(\overleftarrow{i,j},k)$

Let us have 2×2 matrix

$$\mathbf{A} = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix}$$

There exists 2×2 unimodular matrix $\mathbf{E_x}$ so that

$$\mathbf{AE_x} = \begin{pmatrix} g & 0 \\ B_{2,1} & B_{2,2} \end{pmatrix}$$

and

$$\mathbf{E_x} = \begin{pmatrix} p & -A_{1,2}/g \\ q & A_{1,1}/g \end{pmatrix}$$

g is a great common divisor of $A_{1,1}$, $A_{1,2}$ and p, q are numbers satisfying $A_{1,1}p + A_{1,2}q = g$. Unimodular operation represented by matrix $\mathbf{E_x}$ plays an important role both in theory and computation. By embedding the 2×2 matrix $\mathbf{E_x}$ in to the unit matrix $\mathbf{I}$ we obtain an operation which can nullify arbitrary element in matrix $\mathbf{A}$ using some element in the same row. This is illustrated in Figure 2. We call this kind of operation *Euclidean* and denote it by $\mathbf{C_E}(\overrightarrow{i,j},k)$ resp. $\mathbf{C_E}(\overleftarrow{i,j},k)$ where $\mathbf{C_E}(\overrightarrow{i,j},k)$ resp. $\mathbf{C_E}(\overleftarrow{i,j},k)$ nullifies element $A_{k,j}$ using element $A_{k,i}$ resp. element $A_{k,i}$ using element $A_{k,j}$.

Let us suppose that there is some cyclic process of elementary operations e.g. $\mathbf{A^c}(\overrightarrow{i_1,i_2},k_1)$, $\mathbf{A^c}(\overrightarrow{i_2,i_3},k_2)$, $\mathbf{A^c}(\overleftarrow{i_1,i_3},k_3)$. We refer to such process as to a *folding resp. unfolding* if there is an increasing resp. decreasing tendency in matrix entries.

$$
\begin{array}{c|c}
 & \begin{array}{cccccccc}
a_{1,1} & a_{1,2} & a_{1,3} & \cdot & \cdot & \cdot & & \\
a_{2,1} & & & & & & & \\
a_{3,1} & & & & & & & \\
\cdot & \cdot & \cdot & & a_{k,i} & \cdot & a_{k,j} & \\
\cdot & & & & \cdot & & \cdot & \\
\cdot & & & & \cdot & & \cdot & \\
 & & & & \cdot & & \cdot & \\
 & & & & \cdot & & \cdot & a_{n,n}
\end{array} \\
\hline
 & i \rightarrow j
\end{array}
\quad
\begin{array}{c|c}
 & \begin{array}{cccccccc}
1 & & & & & & & \\
 & 1 & & & & & & \\
 & & 1 & & & & & \\
 & & & 1 & & & & \\
\cdot & \cdot & \cdot & \cdot & p & \cdot & -a_{k,j}/g & \\
 & & & & \cdot & 1 & \cdot & \\
\cdot & \cdot & \cdot & \cdot & q & \cdot & a_{k,i}/g & \\
 & & & & \cdot & & \cdot & 1
\end{array} \\
\hline
 & i \qquad j
\end{array}
$$

Row labels: $k \cdots$ (left matrix, fourth row); $i \cdots$ and $j \cdots$ (right matrix, fifth and seventh rows).

Figure 2:

Euclidean operation $\mathbf{C_E}(\overrightarrow{i,j},k)$ in matrix form. When a matrix $\mathbf{A}$ is multiplied by an Euclidean operation matrix the resulting matrix will have $g = \gcd(a_{k,i}, a_{k,j})$ at the place (k,i) and zero at the place (k,j).

1.2 Algorithms for HNF and SNF

The computational aspects of Hermite normal form and Smith normal form begun to be more widely investigated about 1950. In 1952 Rosser [4] proposed an algorithm to compute the Hermite normal form using only elementary operations (see Definition 1).

D. A. Smith [5] and Hu [8] describe an algorithm which follows a constructive proof of Theorem 3 as is in Newman [13]. The algorithm reduces the first row, then proceeds to the first column and if the elements in the first row become to be nonzero the process is repeated. Finally, the first row and column equals zero except the element on the diagonal. Algorithm proceeds to the second row and the second column and so on.

Early it was seen that attention must be given to the phenomenon associated: when unfolding operations occur in some row of the matrix (leading to the gcd of the row resp. column elements) a folding occurs at the same time in other rows of the matrix. Entries came to be very large. This problem was explicitly formulated by several authors and was called expression swell [16] or entry explosion [19].

$$\begin{pmatrix} * & 1 & 2 & 3 & 4 & 5 \\ 16 & * & 6 & 7 & 8 & 9 \\ 17 & 18 & * & 10 & 11 & 12 \\ 19 & 20 & 21 & * & 13 & 14 \\ 22 & 23 & 24 & 25 & * & 15 \\ 26 & 27 & 28 & 29 & 30 & * \end{pmatrix}$$

Figure 3:

The order of computation in Bradley algorithm (simplified form); for example number 5 means the entry is computed at a fith stage of computation. Matrix entries above the diagonal are nullified in row order and entries under the diagonal are reduced modulo diagonal elements also in row order.

The next important step was made by Bradley in [11] where he introduced the operation which we call Euclidean (see Figure 2). This operation allows to nullify an arbitrary element in a matrix with one matrix multiplication. The order of computation in Bradley process for Hermite normal form is in simplified form illustrated in Figure 3. From that time on all proposed algorithms use Euclidean operations as a basic tool and the main attention is given to the analysis of expression swell even if the algorithm of Rosser is experimentally investigated (see for example [25, 31]). But the theoretical analysis remains open and there are no bounds proved on algorithms using only elementary operations. Some notes can be found in [21] and [9].

Kannan and Bachem proposed the first algorithms with a known polynomial expression swell in 1979 [20]. The order of computation of Hermite normal form is illustrated in Figure 4. They proved the upper bound on the number of bits for the largest matrix entry to be $n^3(\log_2(n)+\log_2 \mid\mid \mathbf{A} \mid\mid)$. The polynomial algorithm they have proposed for Smith normal form relies on successive computations of Hermite normal form and so called Lower Hermite normal form. The Lower Hermite normal form is a Hermite normal form of a transposed matrix. They prove that this process is convergent to the Smith normal form of the input matrix.

Chou and Collins in 1982 [21] showed that a better upper bound on the number of bits can be proved on the same algorithm of Kannan and Bachem; namely the $n^2(\log_2(n) + \log_2 \mid\mid \mathbf{A} \mid\mid)$. Then they introduced a new algorithm

$$\begin{pmatrix} * & 1 & 3 & 7 & 13 & 21 \\ 2 & * & 4 & 8 & 14 & 22 \\ 5 & 6 & * & 9 & 15 & 23 \\ 10 & 11 & 12 & * & 16 & 24 \\ 17 & 18 & 19 & 20 & * & 25 \\ 26 & 27 & 28 & 29 & 30 & * \end{pmatrix}$$

Figure 4:

The order of computation in Kannan, Bachem algorithm. Matrix entries above the diagonal are nullified and entries under the diagonal are reduced modulo diagonal elements. Succesively the $1 \times 1, \ldots, n \times n$ principal submatrices are placed in Hermite normal form.

$$\begin{pmatrix} * & 1 & 2 & 4 & 7 & 11 \\ 26 & * & 3 & 5 & 8 & 12 \\ 27 & 22 & * & 6 & 9 & 13 \\ 28 & 23 & 19 & * & 10 & 14 \\ 29 & 24 & 20 & 17 & * & 15 \\ 30 & 25 & 21 & 18 & 16 & * \end{pmatrix}$$

Figure 5:

The order of computation in the Chou, Collins algorithm. Matrix entries above the diagonal are nullified first in the same order as in the Kannan, Bachem algorithm and then entries under the diagonal are reduced modulo diagonal elements.

for Hermite normal form computation where by reordering the computation in Kannan, Bachem process as is illustrated in Figure 5

they achieved an upper bound of $n(\log_2(n) + \log_2 \|\, \mathbf{A} \,\|)$.

In 1987 Domich, Kannan and Trotter introduced a modulo determinant arithmetic [25]. They proved an upper bound $n(\log_2(n) + \log_2 \|\, \mathbf{A} \,\|)$ on the number of bits for their modular method. The basic strategy can be seen by means of lattices. An lattice of matrix $\mathbf{A}$ is defined to be a set $L(\mathbf{A}) = \{\mathbf{A}x \mid x \text{ is integral}\}$. Then by simple reasoning it can be shown that $\det(\mathbf{A})\mathbf{A}^{-1}$ is an integral matrix so that $\mathbf{A}(\det(\mathbf{A})\mathbf{A}^{-1}) = d\mathbf{I}$ is also integral which leads to the fact that $de_i \in L(\mathbf{A})$. Consequently a determinant can be used to reduce the matrix entries during the computation because the Hermite normal form is a basis for $L(\mathbf{A})$. A careful statement of this new strategy and computer experiments in [25] showed that efficient algorithms can be constructed. A modulo determinant computation has one very important feature, it can be combined with any other method of computing a Hermite normal form of a matrix.

Because Hermite normal form of an integral matrix can be computed with modulo determinant arithmetic also Smith normal form can be computed using modulo determinant arithmetic when we use successive Hermite normal form computations with the method developed in [25]. Similar method of Smith normal form computation is also presented by Iliopoulos in [29, 30]. Iliopoulos has also improved the upper bounds on Hermite normal form and Smith normal form computation. But there is no method how to use modulo determinant computation (or any other modular strategy) when the Smith normal form computation does not rely on successive Hermite normal form computations. To see what is going on let us construct a block matrix with dimension $2n$:

$$\begin{pmatrix} \mathbf{A} & d\mathbf{I} \\ d\mathbf{I} & 0 \end{pmatrix}$$

It is straightforward to see that the Smith normal form of this matrix has as the first n elements the Smith normal form of matrix $\mathbf{A}$. Now performing reductions modulo d (determinant) during the computation means in fact the following pattern when described with unimodular matrices:

$$\begin{pmatrix} \mathbf{L} & \mathbf{L}_1 \\ 0 & \mathbf{R}^{-1} \end{pmatrix} \times \begin{pmatrix} \mathbf{A} & d\mathbf{I} \\ d\mathbf{I} & 0 \end{pmatrix} \times \begin{pmatrix} \mathbf{R} & 0 \\ \mathbf{R}_1 & \mathbf{L}^{-1} \end{pmatrix} = \begin{pmatrix} \mathbf{LAR} + \mathbf{dL_1R_1} & d\mathbf{I} \\ d\mathbf{I} & 0 \end{pmatrix}$$

So we obtain a diagonal matrix $\mathbf{LAR} + \mathbf{dL_1R_1}$ but the accurate relation between this matrix and the Smith normal form of the original matrix $\mathbf{A}$ is generally not clear.

Another method of Hermite normal form and Smith normal form computation consists in residual computation modulo relatively prime factorization of determinant. This is developed in [25]. Computation modulo various prime factors was also considered [23] but the problem is to choose appropriately the prime moduli to obtain correct Smith normal form of a given matrix. The answer is known only for special classes of matrices.

Recently new methods has been developed to compute Smith normal form via randomization technique as is described in [28, 35].

The question if the number of bits for the largest matrix entry is polynomialy bounded in the Bradley algorithms for Hermite normal form and Smith normal form is still open even if some experiments [18] seem to show it is not.

2 Upper triangular unimodular matrices

2.1 Introduction

In this section the basic operations of our computational model are matrix multiplications by elementary operations. We define a notion of complexity of Hermite normal form computation in terms of elementary operations. By elementary operations we strictly mean the operations defined in definition 1. Our approach is constructive through means of upper triangular right unimodular matrices. A unimodular operation can be decomposed to a sequence of elementary operations which is not generally unique. But the question is how many such operations are generally required at least. The aim is to show that there exists a matrix (resp. a sequence of matrices) for which every algorithm computing the Hermite normal form by means of the elementary operations $\mathbf{S^c}(i,j)$, $\mathbf{M^c}(i)$, $\mathbf{A^c}(\overrightarrow{i,j},k)$, $\mathbf{A^c}(\overleftarrow{i,j},k)$ needs at least $\frac{n^2-n}{2}$ of such operations.

The reasoning has two steps. Firstly, we construct a different special upper triangular unimodular matrix sequence to bring more light on the above main statement. This matrix sequence (see Construction 4) consists of matrices having ones at the diagonal and different primes above the diagonal but only $O(n^{\alpha(n)})$ $(\alpha(n) < 2)$ of elementary operations is sufficient to build them. This fact shows that introducing different primes does not lead to the lower bound of a number of elementary operations. But on the other hand it also points to the kind of structures which must be destroyed to obtain the lower bound (see Construction 5).

The consequence of asking questions about upper triangular unimodular matrices is that the complexity analysis of Hermite normal form is split into two parts. One is strictly related to the matrix structure and not to the structure of the Euclidean ring of integers itself. The former problems can be studied on upper triangular unimodular matrices. Other questions arise when the general unimodular matrix is taken into account because elements on both sides of the diagonal indicate Euclidean iterations. This produces an extremely complicated mixture of Euclidean algorithm complexity questions and matrix flow complexity questions.

2.2 Upper triangular unimodular matrix construction

The aim of this section is to show that introducing in every step of a computation a new different prime does not lead to the worst case. A contradictory sequence of special integer matrices consisting from different primes is constructed.

The construction is based on the twin primes. There is a well known conjecture that infinitely many such pairs of primes exist [3]. Even if this is not true sufficiently large matrices can be constructed to illustrate some data flow phenomena in integer matrix computations. Let us suppose that all primes are reordered to two sequences:

$$P_A = ((3,5),(11,13),(17,19),\ldots,(a_n^L,a_n^R),\ldots) \text{ and } P_B = (b_1,b_2,b_3,\ldots)$$

sequence P_A contains all pairs of twin primes in increasing order and sequence P_B contains all other primes in increasing order. During the construction we gather successively from both sequences so that two pointers I_{P_A} and I_{P_B} must be maintained to know from what point in a sequence unused primes can be taken. The dimension of matrices is $n = 2^k$ where $k \geq 2$. In fact we construct a sequence of elementary operations. When they are applied to unit matrix $\mathbf{I}$ an upper triangular unimodular matrix is obtained.

For convenience both column and row operations are used; a row operation means placing the corresponding column operation at the beginning of the sequence while a column operation means placing the operation at the end of the sequence. For $\mathbf{S^r}(i,j)$, $\mathbf{M^r}(i)$, $\mathbf{A^r}(\overrightarrow{i,j},k)$ resp. $\mathbf{A^r}(\overleftarrow{i,j},k)$ corresponding columnar operations are $\mathbf{S^c}(i,j)$, $\mathbf{M^c}(i)$, $\mathbf{A^c}(\overleftarrow{i,j},k)$ resp. $\mathbf{A^c}(\overrightarrow{i,j},k)$.

Matrices are constructed inductively for $n = 2^k$ starting with $k = 2$. 4×4 matrix for $k = 2$ we construct by following steps:

1. start with an empty elementary operation sequence.
2. take a new prime b from P_B and add b times row 2 to row 1. See Figure 6 .
3. add two times row 4 to row 2, add two times row 4 to row 1. See Figure 6.

$$
\left(\begin{array}{cccccccc|cccc|cccc}
1&\ast&\ast&\ast&\ast&\ast&\ast&\ast&&&&&&&&\\
&1&\ast&\ast&\ast&\ast&\ast&\ast&&&&&&&&\\
&&1&\ast&\ast&\ast&\ast&\ast&&&&&&&&\\
&&&1&\ast&\ast&\ast&\ast&&&&&&&&\\
&&&&1&\ast&\ast&\ast&&&&&&&&\\
&&&&&1&\ast&\ast&&&&&&&&\\
&&&&&&1&\ast&&&&&&&&\\
&&&&&&&1&&&&&&&&\\
\hline
&&&&&&&&1&&&&&&&\\
&&&&&&&&&1&&&&&&\\
&&&&&&&&&&1&&&&&\\
&&&&&&&&&&&1&&&&\\
&&&&&&&&&&&&1&&&\\
&&&&&&&&&&&&&1&&\\
&&&&&&&&&&&&&&1&\\
&&&&&&&&&&&&&&&1
\end{array}\right)
\left(\begin{array}{cccccccc|cccc|cccc}
1&\ast&\ast&\ast&\ast&\ast&\ast&\ast&&&&&2&&&\\
&1&\ast&\ast&\ast&\ast&\ast&\ast&&&&&2&&&\\
&&1&\ast&\ast&\ast&\ast&\ast&&&&&2&&&\\
&&&1&\ast&\ast&\ast&\ast&&&&&2&&&\\
&&&&1&\ast&\ast&\ast&&&&&2&&&\\
&&&&&1&\ast&\ast&&&&&2&&&\\
&&&&&&1&\ast&&&&&2&&&\\
&&&&&&&1&&&&&2&&&\\
\hline
&&&&&&&&1&&&&&&&\\
&&&&&&&&&1&&&&&&\\
&&&&&&&&&&1&&&&&\\
&&&&&&&&&&&1&&&&\\
&&&&&&&&&&&&1&&&\\
&&&&&&&&&&&&&1&&\\
&&&&&&&&&&&&&&1&\\
&&&&&&&&&&&&&&&1
\end{array}\right)
$$

Figure 6:

Illustration of Construction 4. The upper left triangle is recurrently built then 2's are produced by elementary row operations.

4. take a new left twin a^L from P_A and add a^L times row 3 to row 2, repeat this for row 1. See Figure 7.
5. add column 3 to column 4. See Figure 8.
6. nullify one at position (3,4) by subtracting row 4 from row 3. See Figure 8.
7. take a new prime b from P_B and add b times row 4 to row 3. See Figure 8.

An equivalent formulation by means of an elementary operations sequence is:

1. start with an empty elementary operation sequence.
2. Place $\mathbf{A^c}(\overrightarrow{1,2}, b_1)$ at the beginning of the sequence.
3. Place $\mathbf{A^c}(\overrightarrow{1,4}, 2)$, $\mathbf{A^c}(\overrightarrow{2,4}, 2)$ at the beginning of the sequence.
4. Place $\mathbf{A^c}(\overrightarrow{1,3}, a_2^L)$, $\mathbf{A^c}(\overrightarrow{2,3}, a_1^L)$ at the beginning of the sequence.
5. Place $\mathbf{A^c}(\overrightarrow{3,4}, 2))$ at the end of the sequence.
6. Place $\mathbf{A^c}(\overrightarrow{3,4}, -1)$ at the beginning of the sequence.

7. Place $\mathbf{A^c}(\overrightarrow{3,4}, b_2)$ at the beginning of the sequence.

Previous steps correspond to the operation sequence ($\mathbf{A^c}(\overrightarrow{3,4}, b_2)$, $\mathbf{A^c}(\overrightarrow{3,4}, -1)$ $\mathbf{A^c}(\overrightarrow{1,3}, a_2^L)$, $\mathbf{A^c}(\overrightarrow{2,3}, a_1^L)$, $\mathbf{A^c}(\overrightarrow{1,4}, 2)$, $\mathbf{A^c}(\overrightarrow{2,4}, 2)$, $\mathbf{A^c}(\overrightarrow{1,2}, b_1)$, $\mathbf{A^c}(\overrightarrow{3,4}, 2)$)

Now let us suppose that we have a $2^k \times 2^k$ matrix $\mathbf{A}_k$ for some k. The matrix $\mathbf{A}_{k+1}$ will be constructed by the following steps:

1. start with the elementary operation sequence for $\mathbf{A}_k$. Unit matrix $\mathbf{I}$ is multiplied by this sequence so that the $2^k \times 2^k$ principal submatrix of $\mathbf{A}_{k+1}$ equals the matrix $\mathbf{A}_k$. See Figure 6.

2. Produce half a column of 2's by placing the operation subsequence $\mathbf{A^c}(\overrightarrow{1, 2^k + 2^{k-1}}, 2)$, $\ldots$, $\mathbf{A^c}(\overrightarrow{2^k - 1, 2^k + 2^{k-1}}, 2)$, $\mathbf{A^c}(\overrightarrow{2^k, 2^k + 2^{k-1}}, 2)$ at the beginning of the sequence (row operations). See Figure 6.

3. By column operations fill the right rectangle with 2's or equivalently place the operation subsequence
$\mathbf{A^c}(\overrightarrow{2^k + 2^{k-1}, 2^k + 2^{k-1} + 1}, 1)$, $\mathbf{A^c}(\overrightarrow{2^k + 2^{k-1}, 2^k + 2^{k-1} + 2}, 1)$,$\ldots$,
$\mathbf{A^c}(\overrightarrow{2^k + 2^{k-1}, 2^k + 2^{k-1} + 2^{k-1}}, 1)$ at the end of the sequence. See Figure 7.

4. Nullify ones by placing
$\mathbf{A^c}(\overrightarrow{2^k + 2^{k-1}, 2^k + 2^{k-1} + 1}, -1)$, $\ldots$,
$\mathbf{A^c}(\overrightarrow{2^k + 2^{k-1}, 2^k + 2^{k-1} + 2^{k-1}}, -1)$ at the beginning of the sequence (row operations). See Figure 7.

5. Fill the left rectangle with left twins by placing
$\mathbf{A^c}(\overrightarrow{2^k - 1, 2^k}, a_{m_1}^L)$, $\ldots$, $\mathbf{A^c}(\overrightarrow{1, 2^k + 2^{k-1} - 1}, a_{m_2}^L)$ at the beginning of the sequence (row operations). See Figure 7.

6. Produce right twins by column operations or equivalently place the subsequence
$\mathbf{A^c}(\overrightarrow{2^k, 2^k + 2^{k-1}}, 1)$, $\ldots$, $\mathbf{A^c}(\overrightarrow{2^k + 2^{k-1} - 1, 2^k + 2^{k-1} + 2^{k-1}}, 1)$ at the end of the operation sequence. See Figure 8.

$$
\left(\begin{array}{cccccccc|cccc|cccc}
1 & \ast & \ast & \ast & \ast & \ast & \ast & \ast & & & & & 2 & 2 & 2 & 2 \\
 & 1 & \ast & \ast & \ast & \ast & \ast & \ast & & & & & 2 & 2 & 2 & 2 \\
 & & 1 & \ast & \ast & \ast & \ast & \ast & & & & & 2 & 2 & 2 & 2 \\
 & & & 1 & \ast & \ast & \ast & \ast & & & & & 2 & 2 & 2 & 2 \\
 & & & & 1 & \ast & \ast & \ast & & & & & 2 & 2 & 2 & 2 \\
 & & & & & 1 & \ast & \ast & & & & & 2 & 2 & 2 & 2 \\
 & & & & & & 1 & \ast & & & & & 2 & 2 & 2 & 2 \\
 & & & & & & & 1 & & & & & 2 & 2 & 2 & 2 \\
\hline
 & & & & & & & & 1 & & & & & & & \\
 & & & & & & & & & 1 & & & & & & \\
 & & & & & & & & & & 1 & & & & & \\
 & & & & & & & & & & & 1 & & & & \\
 & & & & & & & & & & & & 1 & 1 & 1 & 1 \\
 & & & & & & & & & & & & & 1 & & \\
 & & & & & & & & & & & & & & 1 & \\
 & & & & & & & & & & & & & & & 1
\end{array}\right)
\left(\begin{array}{cccccccc|cccc|cccc}
1 & \ast & \ast & \ast & \ast & \ast & \ast & \ast & L & L & L & L & 2 & 2 & 2 & 2 \\
 & 1 & \ast & \ast & \ast & \ast & \ast & \ast & L & L & L & L & 2 & 2 & 2 & 2 \\
 & & 1 & \ast & \ast & \ast & \ast & \ast & L & L & L & L & 2 & 2 & 2 & 2 \\
 & & & 1 & \ast & \ast & \ast & \ast & L & L & L & L & 2 & 2 & 2 & 2 \\
 & & & & 1 & \ast & \ast & \ast & L & L & L & L & 2 & 2 & 2 & 2 \\
 & & & & & 1 & \ast & \ast & L & L & L & L & 2 & 2 & 2 & 2 \\
 & & & & & & 1 & \ast & L & L & L & L & 2 & 2 & 2 & 2 \\
 & & & & & & & 1 & L & L & L & L & 2 & 2 & 2 & 2 \\
\hline
 & & & & & & & & 1 & & & & & & & \\
 & & & & & & & & & 1 & & & & & & \\
 & & & & & & & & & & 1 & & & & & \\
 & & & & & & & & & & & 1 & & & & \\
 & & & & & & & & & & & & 1 & & & \\
 & & & & & & & & & & & & & 1 & & \\
 & & & & & & & & & & & & & & 1 & \\
 & & & & & & & & & & & & & & & 1
\end{array}\right)
$$

Figure 7:

Illustration of Construction 4. By column operations a rectangle of 2's is built then 1's under this rectangle are nullified by row operations. The neighbouring rectangle is filled with left members of twin prime pairs.

7. Nullify ones under the right rectangle by placing $\mathbf{A^c}(2^k+1, \overrightarrow{2^k}+2^{k-1}, -1), \ldots, \mathbf{A^c}(2^k+2^{k-1}-1, \overrightarrow{2^k}+2^{k-1}+2^{k-1}, -1)$ at the beginning of the sequence (row operations). See Figure 8.

8. Fill the lower right triangle with members of P_B by placing $\mathbf{A^c}(2^k, \overrightarrow{2^k}+1, b_{s_1}), \ldots, \mathbf{A^c}(2^k+\overrightarrow{2^k}-1, 2^{k+1}, b_{s_2})$ at the beginning of the sequence (row operations). See Figure 8, resulting matrix for $k = 4$ is illustrated in Figure 9

To construct a starting matrix $\mathbf{A}_k$ for $k = 2$ eight elementary operations are sufficient. It can be seen that step 2 requires 2^k elementary operations, step 3 requires 2^{k-1} elementary operations, step 4 requires 2^{k-1} elementary operations, step 5 requires 2^{2k-1} elementary operations, step 6 requires 2^{k-1} elementary operations, step 7 requires 2^{k-1} elementary operations and step 8 requires $2^{2k-1} - 2^{k-1}$ elementary operations. A recurrence relation can be derived:

$$g(k+1) = g(k) + 2^{2k} + 5 \times 2^{k-1}$$

$$
\left(\begin{array}{cccccccc|cccc|cccc}
1 & * & * & * & * & * & * & * & L & L & L & L & R & R & R & R \\
 & 1 & * & * & * & * & * & * & L & L & L & L & R & R & R & R \\
 & & 1 & * & * & * & * & * & L & L & L & L & R & R & R & R \\
 & & & 1 & * & * & * & * & L & L & L & L & R & R & R & R \\
 & & & & 1 & * & * & * & L & L & L & L & R & R & R & R \\
 & & & & & 1 & * & * & L & L & L & L & R & R & R & R \\
 & & & & & & 1 & * & L & L & L & L & R & R & R & R \\
 & & & & & & & 1 & L & L & L & L & R & R & R & R \\
\hline
 & & & & & & & & 1 & & & & 1 & & & \\
 & & & & & & & & & 1 & & & & 1 & & \\
 & & & & & & & & & & 1 & & & & 1 & \\
 & & & & & & & & & & & 1 & & & & 1 \\
 & & & & & & & & & & & & 1 & & & \\
 & & & & & & & & & & & & & 1 & & \\
 & & & & & & & & & & & & & & 1 & \\
 & & & & & & & & & & & & & & & 1
\end{array}\right)
$$

$$
\left(\begin{array}{cccccccc|cccc|cccc}
1 & * & * & * & * & * & * & * & L & L & L & L & R & R & R & R \\
 & 1 & * & * & * & * & * & * & L & L & L & L & R & R & R & R \\
 & & 1 & * & * & * & * & * & L & L & L & L & R & R & R & R \\
 & & & 1 & * & * & * & * & L & L & L & L & R & R & R & R \\
 & & & & 1 & * & * & * & L & L & L & L & R & R & R & R \\
 & & & & & 1 & * & * & L & L & L & L & R & R & R & R \\
 & & & & & & 1 & * & L & L & L & L & R & R & R & R \\
 & & & & & & & 1 & L & L & L & L & R & R & R & R \\
\hline
 & & & & & & & & 1 & b & b & b & b & b & b & b \\
 & & & & & & & & & 1 & b & b & b & b & b & b \\
 & & & & & & & & & & 1 & b & b & b & b & b \\
 & & & & & & & & & & & 1 & b & b & b & b \\
 & & & & & & & & & & & & 1 & b & b & b \\
 & & & & & & & & & & & & & 1 & b & b \\
 & & & & & & & & & & & & & & 1 & b \\
 & & & & & & & & & & & & & & & 1
\end{array}\right)
$$

Figure 8:

Illustration of Construction 4. Columns of left twins are added to columns of 2's producing right twins then 1's under the right rectangle are nullified by row operations. Finally the lower right triangle is filled with members of P_B sequnce.

$$\left(\begin{array}{cc|c|c|cc|cc|cccc|cccc}
1 & b & L & R & L & L & R & R & L & L & L & L & R & R & R & R \\
 & 1 & L & R & L & L & R & R & L & L & L & L & R & R & R & R \\
\hline
 & & 1 & b & L & L & R & R & L & L & L & L & R & R & R & R \\
 & & & 1 & L & L & R & R & L & L & L & L & R & R & R & R \\
\hline
 & & & & 1 & b & b & b & L & L & L & L & R & R & R & R \\
 & & & & & 1 & b & b & L & L & L & L & R & R & R & R \\
 & & & & & & 1 & b & L & L & L & L & R & R & R & R \\
 & & & & & & & 1 & L & L & L & L & R & R & R & R \\
\hline
 & & & & & & & & 1 & b & b & b & b & b & b & b \\
 & & & & & & & & & 1 & b & b & b & b & b & b \\
 & & & & & & & & & & 1 & b & b & b & b & b \\
 & & & & & & & & & & & 1 & b & b & b & b \\
 & & & & & & & & & & & & 1 & b & b & b \\
 & & & & & & & & & & & & & 1 & b & b \\
 & & & & & & & & & & & & & & 1 & b \\
 & & & & & & & & & & & & & & & 1
\end{array}\right)$$

Figure 9:

Illustration of Construction 4.

where $g(k)$ is the number of elementary operations sufficient for the matrix $\mathbf{A}_k$ with dimension $n = 2^k$. The fastest growing term is 2^{2k} so we can write:

$$g(k) = \sum_{i=0}^{k-1} 2^{2i} + f(k) \quad \text{and} \quad g(k) = 2^{\alpha(k)(k)} + f(k).$$

where $\alpha(k) < 2$ because $3\sum_{i=0}^{k-1} 2^{2i} + 1 = 2^{2k}$ so $\sum_{i=0}^{k-1} 2^{2i} \leq 2^{2k}/3$ and $2^{\alpha k} = 2^{2k}/3$ so $\alpha = 2 - \log_2 3/k$. We can summarize:

Construction 4 *There exists a sufficiently long sequence of upper triangular unimodular matrices $\mathbf{A}_k$ having different primes above the diagonal with dimension $n = 2^k$ for which only $O(n^{\alpha(n)})$ $(\alpha(n) < 2)$ of the elementary operations is sufficient to build the matrix.*

2.3 The lower bound

The previous construction has shown that introducing different primes is not the right way to construct the worst matrix but has also shown some way to do it. Firstly, let us state the goal:

Construction 5 [1] *For every n there exists an upper triangular unimodular matrix $\mathbf{R}_W$ with dimension n for which every decomposition to elementary operations has at least $\frac{n^2-n}{2}$ of elementary operations.*

[1] The construction is now under development. In the case of an interest about the current state please contact the authors.

3 Gcd networks

3.1 Introduction

Finding the greatest common divisor (gcd) of integers is a basic mental step in the analysis of numbers therefore great effort was invested in algorithmic treatment of this problem. The existence of computing machines has brought special attention to this area, which is renewed every time a substantially new architecture of computing machines is invented.

Now massively parallel computers are under explosive development and investigations of their possible structure and algorithms. So the question of computing the gcd of n numbers is once more investigated. There are many articles about algorithms for sequential machines and their analysis (for summary see[12, 27]) even if the problem of computing gcd of n numbers is not treated explicitly (the most natural reduction to the case of the gcd of two numbers was analysed by Bradley in [9]). Our investigations are directed more towards algorithms intended for machines generally covered with the term SIMDs than to MIMDs. But some ideas could be worthwhile for the MIMDs as well.

The basic working set of ideas we move in could be determined by the following terms: permutation networks, comparison networks and sorting networks. The crucial problem for algorithms design here is to choose a good atomization (resp. granularity) of arithmetic operations. We use the terms atomization and granularity quite freely here. By atomization we mean (intuitively) how the problem is divided into parts which are the basic building bricks of a solution (from the context it is clear if we suppose a mapping between the atoms and processing elements). With the word granularity we mean the complexity of atoms in relation to the basic universe we move in. Regardless, there must be an elementary operation of branch according to an order of numbers (i.e. comparison operation). Generally, the gcd computation of n numbers can be atomized using the following granularity:

1. the gcd operation on two numbers.
2. the division $x \bmod y$.
3. the subtraction operation.

4. the lowest level operations i.e. the bitwise boolean operations.

Naturally, the above atomization is related at the lowest level to the architecture and number system of todays computing machines in use. Neither can we exclude substantially different architectures leading to a different lowest level granularity of atomization nor can we exclude new discoveries in number theory leading to different atomization at the higher levels.

Computer experiments and our current understanding of the problem have led us to focus on an intermediate granularity i.e. we use the division operation together with a comparison operation as an atom.

The greater atoms with type 1 granularity are not very well suited for SIMDs. A gcd operation for two numbers has a character of an event because we do not know in advance how many iterations it will need. Therefore the atomization based on a gcd operation is more suitable for MIMD machines because a synchronization with events is natural for these architectures and processing elements can do some work on different tasks while waiting for the result of a gcd computation. Several kinds of algorithms for the gcd computations on MIMDs are reported [32].

The situation with a higher granularity (type 3, type 4) than the one we have chosen is more complicated. It can be argued in favor of smaller atoms that they improve the suitability of an algorithm for SIMD architectures. However, computational experience shows that the subtraction operation as an atom leads to worse results. The question if even higher granularity (i.e. the use of the lowest level operations) could lead to better results is left completely open but we suggest that for SIMD machines it is not true because of an extreme load of the communication network in this case. It is possible that systolic array architecture could be considered for this case.

The first sublinear (with respect to the number of bits of input numbers) parallel algorithm working on type 4 granularity is due to Kannan [26]. A further improvement was developed in [33]. At the end, there is a very hard question whether gcd problem is in the NC [24] class or not.

For every atomization considered above there is a question if the branch resp. the comparison operation have to form a different atom from the arithmetic operation leading to heterogenous networks or if they have to be combined in one atom. Generally, the design and analysis of heterogenous networks is much more complicated.

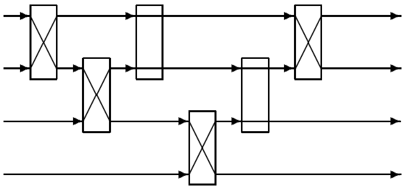


Figure 10:

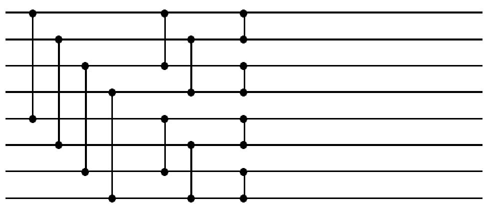
Every framebox represents a switch module. The switch module can either exchange the numbers or pass them in the same order as they are on the input.

In the following parts we firstly define several well known structures i.e. permutation networks, comparison networks and sorting networks. Finally we define two new structures called a *division network* and a *gcd network* which we use in sequel to design algorithms for the parallel gcd computation of n numbers.

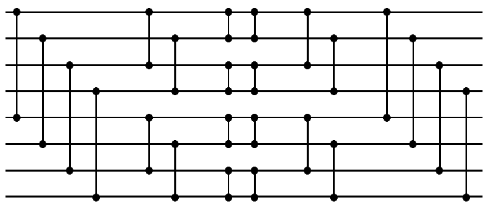
3.2 Permutation networks

A permutation network is a scheme representing a permutation. Input to the scheme is a n-tuple of integers $(a_1, a_2, \ldots, a_n)$ and output are the same numbers but in permuted order. One way to imagine such a network is in Figure 10 where every framebox represents a switch module. The switch modules either exchange the numbers on input or not. This depends on the module state and not on the numbers passed. An input of the scheme is on the left side where we suppose one input number on each line. Usually permutation networks are recursively built from the smaller permutation networks as for example the permutation network in Figure 10 which is built from elementary permutation networks.

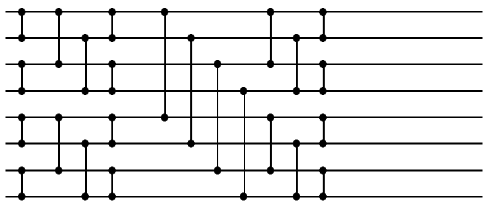
Sometimes the term permutation network is used in a more restrictive way i.e. some authors call a permutation network a network capable of implementing all possible permutations by means of various switch settings. But as we



$L_3 \in \mathcal{L}$



$O_3 \in \Omega$



$B_3 \in \Sigma$

Figure 11:

Various permutation networks. They will be used as basic structures in sequel.

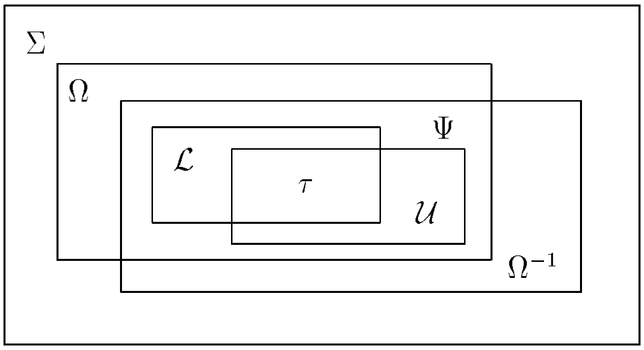


Figure 12:

The classification of permutations from [34].

have defined it above, for us the permutation network is a network together with one instance of switch settings; it is a network representing one concrete permutation.

Permutations (i.e. permutation networks) can be classified in several ways. We use the classification from [34]. The $\mathcal{L}$, $\mathcal{U}$, Ω, Ω^{-1} and Σ classes serve us as basic sources of network structures for further investigations. The relations between these permutation sets are in Figure 12.

Sometimes it is necessary to connect not adjacent lines with a switch module. In Figure 11 is a different way how to represent permutation networks. The representants of classes $\mathcal{L}$, Ω, and Σ are in Figure 11 denoted as L_8, O_8 and B_8. The switch states are not emphasized for two reasons. Firstly, every set of switch states in L_8 produces an $\mathcal{L}$ class permutation network. Secondly, we are not interested in particular switch settings. In further investigations only a position of the switches is used.

3.3 Comparison networks

A comparison network is a scheme similar to the permutation network but instead of the switch modules we have comparison modules. A comparison network is in Figure 13 where every framebox represents a comparison module. The comparison modules either exchange the numbers on input or not

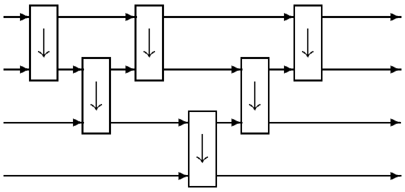


Figure 13:

Every framebox represents a comparatory module. The comparatory modules either exchange the numbers on input or not depending on the condition that the larger number have to be placed in the direction of an arrow.

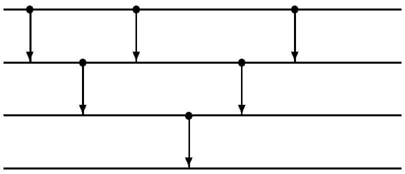


Figure 14:

In the picture the numbers are advancing from the left; comparatory modules between lines are represented by arrows. The comparators perform an exchange of input numbers to obtain an output where a larger number is in the line labeled with arrow.

depending on the input numbers so that a larger number is placed in the direction of an arrow. A different way is used in Figure 14 to represent the same network than in Figure 13.

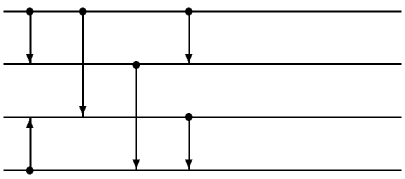


Figure 15:

The sorting network [14] for four numbers.

3.4 Sorting networks

A sorting network is a special kind of a comparison network. Input to the network is a n-tuple of integers $(a_1, a_2, \ldots, a_n)$ and output are the same numbers sorted in ascending or descending order. One such network which sorts four numbers is in Figure 13 where every framebox represents a comparison module. This network sorts four numbers so that on the output the lowest line contains the largest number. A different sorting network is in Figure 15 which cannot be effectively represented in the way used in Figure 13.

There is a well established theory of sorting networks (for the reference see [14]) and they are widely applied in the development of algorithms for modern parallel machines. In the theory two main groups of questions are still open: firstly, there is a question to what extent a number of comparison modules can be minimized. Secondly, when one looks at the Figures 14 and 15 it can be seen that some modules can work in parallel. The minimal number of time steps for the network in Figure 14 is five and for the network in Figure 15 is three. The latter group of questions address the problem of finding networks with minimal delay.

The basic comparison and sorting networks we use in sequel are in Figure 16. They are derived from the basic permutation networks in Figure 11. When we say that a comparison network is of class $\mathcal{L}$ we mean that if comparison modules are replaced with switch modules we obtain a permutation network of that class.

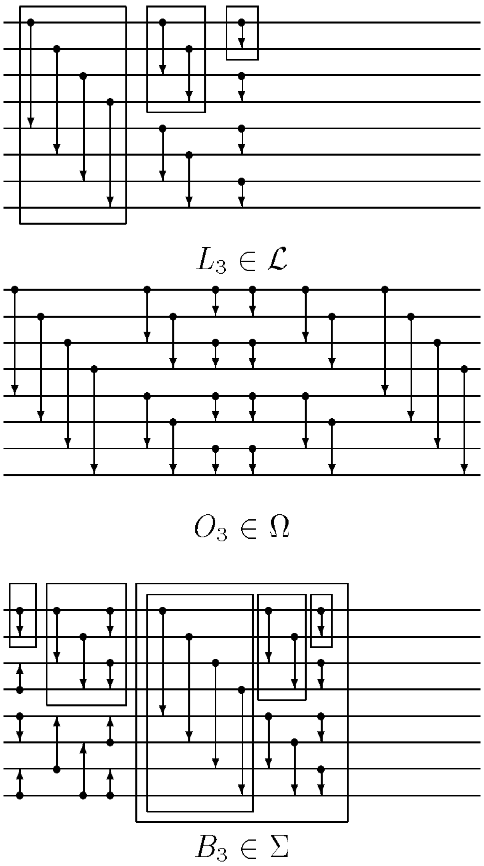


Figure 16:

The comparision networks. The L_3 comparision network is called a bitonic sorter and it consists of seven subnetworks called half-cleaners. The sorting network B_3 is according to [14]. It consists of seven subnetworks-bitonic sorters.

3.5 A division and gcd networks

In this section we introduce two new kinds of networks i.e. a *division network* and a *gcd network*. Without a significant loss of generality we can work only with positive integers. The generalization to all integers is straightforward.

To define a *division network* we firstly define a *division module* as a block with two inputs and two outputs. To draw a division module in figures we use an arrow similarly as in the case of comparison networks (see Figure 13).

The division module firstly compares the input numbers leading a smaller number placement in the direction of the arrow. Then it computes a division of the larger number modulo the smaller number so that the smaller result (the remainder) is placed on the line pointed to by the tail of the arrow. It means that a division module encapsulates two operation. The first operation is a comparison operation and the second is a division operation. When one of the inputs is equal to zero we consider the remainder of the division operation to be zero.

We say that a division network *reduces* an input sequence iff after a finite number of iterations (passes) through the network we have on the output only one non-zero element equal to the gcd of input numbers. By iterations or passes we mean that the output sequence from the network is placed at the input.

We call a division network which reduces all possible input sequences a *gcd network*. For example in Figure 16 we have three gcd networks derived from the $\mathcal{L}$, Ω and Σ class permutation networks. In the following sections we derive a simple criterion to see that these networks are in fact gcd networks.

3.6 A zero-one principle for gcd networks

A zero-one principle is a known method for analysing comparison networks. The zero-one principle gives a sufficient condition to conclude that the particular comparison network is a sorting network. A similar method can be used to prove a division network is a gcd network.

Firstly, let us develop a formalism we use to describe and analyse division networks. We can use as an input to the network a vector $(a_1, a_2, \ldots, a_n)$ of integers (see Figure 17). The action of every division module can be described by a right multiplication with a unimodular matrix. For a division module we

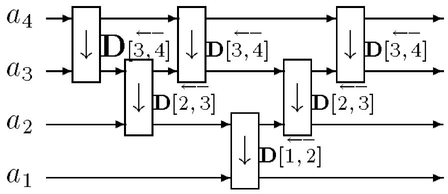


Figure 17:

Every framebox represents a division module.

use a notation $\mathbf{D}[\overrightarrow{i,j}]$ or $\mathbf{D}[\overleftarrow{i,j}]$ where we always suppose $i < j$. The division module $\mathbf{D}[\overrightarrow{i,j}]$ operates between lines i and j in the division network and the arrow points to the line on which a greater result is placed. We denote a division network by a sequence of the division modules

$$(\mathbf{D}[\overrightarrow{i_1,j_1}], \langle \mathbf{D}[\overleftarrow{i_2,j_2}], \mathbf{D}[\overrightarrow{i_3,j_3}] \rangle, \ldots, \mathbf{D}[\overrightarrow{i_n,j_n}])$$

where the brackets means that these modules can act in parallel and we read the modules from left to right in a division network. We call a pass of a sequence of numbers through a maximal set of parallel working modules a *step*. The action of the division module can have only two forms. Either there is an exchange before division or not. These possibilities can be expressed as unimodular operations

1. $\mathbf{D}[\overleftarrow{i,j}]$ acts as $\mathbf{S^c}(i,j)\mathbf{A^c}(\overrightarrow{i,j},k)$
2. $\mathbf{D}[\overleftarrow{i,j}]$ acts as $\mathbf{A^c}(\overrightarrow{i,j},k)$

or in the matrix form

$$\begin{array}{c}\begin{pmatrix} 1 & & \cdot & & & & & \\ & 1 & \cdot & & & & & \\ \cdot & \cdot & 0 & \cdot & \cdot & 1 & & \\ & & \cdot & 1 & & \cdot & & \\ & & \cdot & & 1 & \cdot & & \\ & & 1 & \cdot & \cdot & 0 & & \\ & & \cdot & & & \cdot & 1 & \\ & & \cdot & & & \cdot & & 1 \end{pmatrix} \\ \begin{matrix} & & i & & & j & & \end{matrix}\end{array} \times \begin{array}{c}\begin{pmatrix} 1 & & & & & & & \\ & 1 & & & & & & \\ & & 1 & \cdot & \cdot & k & & \\ & & \cdot & 1 & & \cdot & & \\ & & \cdot & & 1 & \cdot & & \\ & & \cdot & & & 1 & & \\ & & \cdot & & & \cdot & 1 & \\ & & \cdot & & & \cdot & & 1 \end{pmatrix} \\ \begin{matrix} & & i & & \rightarrow & j & & \end{matrix}\end{array} = \begin{array}{c}\begin{pmatrix} 1 & & \cdot & & & & & \\ & 1 & \cdot & & & & & \\ \cdot & \cdot & 0 & \cdot & \cdot & 1 & & \\ & & \cdot & 1 & & \cdot & & \\ & & \cdot & & 1 & \cdot & & \\ & & 1 & \cdot & \cdot & k & & \\ & & \cdot & & & \cdot & 1 & \\ & & \cdot & & & \cdot & & 1 \end{pmatrix} \\ \begin{matrix} & & i & & & j & & \end{matrix}\end{array}$$

or

$$\begin{array}{c}\begin{pmatrix} 1 & & & & & & & \\ & 1 & & & & & & \\ & & 1 & \cdot & \cdot & k & & \\ & & \cdot & 1 & & \cdot & & \\ & & \cdot & & 1 & \cdot & & \\ & & \cdot & & & 1 & & \\ & & \cdot & & & \cdot & 1 & \\ & & \cdot & & & \cdot & & 1 \end{pmatrix} \\ \begin{matrix} & & i & & \rightarrow & j & & \end{matrix}\end{array}$$

Every pass (iteration) of some integer sample $\mathbf{s} = (a_1, a_2, \ldots, a_n)$ through a division network can be expressed as a multiplication of $\mathbf{s}$ by a unimodular matrix. The following definitions are used in sequel:

Definition 6 (Fibonacci matrix for a division network) *Let us have a division network N of order n described by a sequence of division modules:*

$$(\mathbf{D}[\overrightarrow{i_1, j_1}], \mathbf{D}[\overleftarrow{i_2, j_2}], \mathbf{D}[\overrightarrow{i_3, j_3}], \ldots, \mathbf{D}[\overrightarrow{i_n, j_n}])$$

A $n \times n$ unimodular integer matrix $\mathbf{F_N}$ defined by the following procedure we call a Fibonacci matrix of the division network N:

an elementary matrix (operation) $\mathbf{A^c}(\overrightarrow{i, j}, 1)$ (resp. $\mathbf{A^c}(\overleftarrow{i, j}, 1)$) is assigned to every division module $\mathbf{D}[\overleftarrow{i, j}]$ (resp. $\mathbf{D}[\overrightarrow{i, j}]$) in the sequence but in reversed order starting from the tail of the sequence; then the identity matrix is multiplied by this sequence.

For example a network L_3 in Figure 16 has the following fibonacci matrix:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

.

Definition 7 (Subnetwork) *Let us have a division network N defined by the sequence of division modules $(D_1, D_2, \ldots, D_n)$. Then the division network N_s as defined by a subsequence $(D_{i_1}, D_{i_2}, \ldots, D_{i_k})$ is called a division subnetwork of the network N.*

Lemma 8 (Zero-one principle for gcd networks) *If a division network has a subnetwork which reduces all $\binom{n}{2}$ possible sequences of 0's and 1's containing exactly two 1's in one iteration, than it reduces all possible sequences of integers.*

Proof. We denote by $\|x\|_a$ the norm of vector $\mathbf{a}$ defined as

$$\|\mathbf{a}\|_a = \sum_{i=1}^{n} |a_i|$$

Let $\mathbf{a}^i$ means $i-th$ iteration of initial vector $\mathbf{a}^0$. It is clear that if after every pass the norm strictly decreases i.e. $\|\mathbf{a}^i\|_a > \|\mathbf{a}^{(i+1)}\|_a$ then after a finite number of passes there will be a gcd of input numbers as the only nonzero element. Therefore it is sufficient to show that under our assumptions there must be a decrease of norm in every pass through the network if on the input there is a vector with at least two nonzero elements.

Let N_s is a subnetwork of N with assumed property and let us have an arbitrary vector $\mathbf{a}$ on the input of N. Let the vector $\mathbf{a}$ have at least two nonzero elements a_i, a_j. If the norm of $\mathbf{a}$ was reduced by N before N_s there is nothing to prove; if this is not the case then we have a vector $\mathbf{a}'$ with at least two nonzero elements a'_i, a'_j on the input of N_s. But a pass through N_s necessarily reduces one of these two numbers. $\boxed{\text{QED}}$

By a similar reasoning the following variant of the zero-one principle can be also derived

Lemma 9 (Zero-one principle for gcd networks) *If a division network reduces all $\binom{n}{2}$ possible sequences of 0's and 1's containing exactly two 1's than it reduces all possible sequences of integers.*

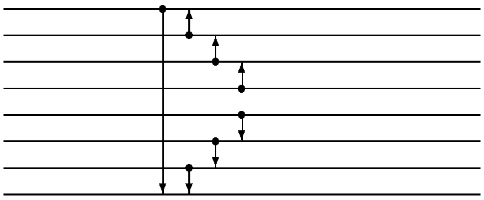


Figure 18:

A gcd network which does not reduce every sequence in one pass.

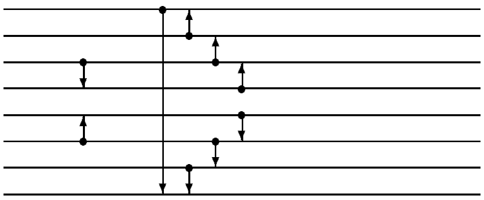


Figure 19:

A division network with a subnetnetwork from Figure 18 which is not a gcd network.

Following the previous lemma we note that the condition in lemma 8 is only sufficient condition not necessary as can be seen in Figure 18 where we have a gcd network which does not reduce every vector of 0's and 1' in one pass. On the other hand when we use only the feature " to reduce" instead of " to reduce in one pass" as it is in lemma 9 we cannot argue with subnetworks as can be seen in Figure 19 where we have a network with a subnetwork from Figure 18. But this is not a gcd network at all. One can argue that a notion of subnetwork is of no need as it is in the case of sorting networks but we suggest that the situation with gcd networks is different and a notion of subnetwork can be worthwhile.

Let us define more accurately now the three basic division networks L_3, O_3 and B_3 in Figure 16. In fact the networks in the Figure are instances of well defined classes of networks. The number of lines for a particular network is sometimes called an order of a network. For convenience, when we work with these classes of networks throughout we assume networks of order 2^i, $i \geq 1$. The division network L_i of an order 2^i is derived from a known comparison network - bitonic sorter. It holds that the first half-cleaner connects lines with numbers differing in $i - th$ bit when we count lines starting from zero. The next half-cleaners connect lines differing in $(i-1)-th$ bit and so on. When we take into the account this fact together with Lemma 8 it is clear that division network L_i is a gcd network for $i \geq 1$. The O_i division network of an order 2^i consists of L_i network together with the same network in reversed order. It is this network we have chosen to investigate as a representation of Ω class structure. From Lemma 8 it is a gcd network. Finally, the division network B_i of an order 2^i is derived from a sorting network. Because it contains a L_i network as a subnetwork it is a gcd network.

3.7 Computational experience

The computational experiments we perform have the following main objectives related to the investigation of parallelism in Hermite normal form and Smith normal form of integral matrices. Firstly, we want to establish appropriate granularity for the parallelism mainly with respect to SIMD architectures. Next aim is to experiment with known methods and compare them with gcd processes realized by gcd networks. Another objective is to deal with various aspects of expression swell. This section describes very shortly one part of experiments. We are determined to prepare a more consistent description of the experimental work in future.

The following table resumes a part of experiments we have done on gcd networks. For the standard Bradley process we count the number of division steps, for parallel networks we count the number of parallel steps times the number of divisions in a module. The table has two parts for every network denoted by "(1)" and "(n)". Denotation "(1)" means the basic module does 1 division operation together with comparison (this is what we have defined as a division block). Denotation "(n)" means the basic module does so many Euclidean iterations that only one pass through the network is sufficient (in fact we did experiments with an increasing number of division operations in the module).

The main tool for these experiments is the MAPLE language. We have collected in the table mean values for 10 random samples. As a random generator we use a standard MAPLE generator.

	Order of network					
	4 (1)	16 (1)	64 (1)	4 (23)	16 (28)	64 (29)
standard gcd Bradley process	26	50	145			
L_i network	22	24	30	46	112	348
O_i network	28	24	36	92	224	348
B_i network	32	48	72	92	448	1044

4 Pseudocommutativity

4.1 Introduction

When a sequence of elementary operations is defined one can ask how this sequence can be equivalently reordered or what consequences some reordering could have. We look for relations of a pseudocommutativity of the following kind:

- $\mathbf{S^c}(r,s)\mathbf{A^c}(\overrightarrow{i,j},k) = \mathbf{A^c}(\overrightarrow{.,.},.)\mathbf{S^c}(.,.)$
- $\mathbf{M^c}(l)\mathbf{A^c}(\overrightarrow{i,j},k) = \mathbf{A^c}(\overrightarrow{.,.},.)\mathbf{M^c}(.)$
- $\mathbf{M^c}(l)\mathbf{S^c}(i,j) = \mathbf{S^c}(.,.)\mathbf{M^c}(.)$

Two unitary operations are called *disjoint* when they operate on disjoint sets of column resp. row indices. We also introduce a formal operation *reverse arrow*:

$$\mathbf{R_a}(\mathbf{A^c}(\overrightarrow{i,j},k)) = \mathbf{A^c}(\overleftarrow{i,j},k)$$

As was emphasized in Def. 1 in $\mathbf{S^c}(i,j)$ and in $\mathbf{A^c}(\overrightarrow{i,j},k)$ always $i < j$ is assumed. By simple reasoning it can be shown that

Lemma 10 (S - M commutation, swap-multiply commutation)
$\mathbf{S}\ \mathbf{M} = \mathbf{M}\ \mathbf{S}$ *when they are disjoint resp.*
$\mathbf{S^c}(i,j)\mathbf{M^c}(k) = \mathbf{M^c}(k)\mathbf{S^c}(i,j)$ *when* $k \neq i,j$ *and*

$\mathbf{S^c}(i,j)\mathbf{M^c}(k) = \mathbf{M^c}(j)\mathbf{S^c}(i,j)$ *when* $k = i$
$\mathbf{S^c}(i,j)\mathbf{M^c}(k) = \mathbf{M^c}(i)\mathbf{S^c}(i,j)$ *when* $k = j$

Lemma 11 (A - M commutation, add-multiply commutation)
$\mathbf{A}\ \mathbf{M} = \mathbf{M}\ \mathbf{A}$ *when they are disjoint resp.*
$\mathbf{A^c}(\overrightarrow{i,j},k)\mathbf{M^c}(l) = \mathbf{M^c}(l)\mathbf{A^c}(\overrightarrow{i,j},k)$ *when* $l \neq i,j$
$\mathbf{A^c}(\overleftarrow{i,j},k)\mathbf{M^c}(l) = \mathbf{M^c}(l)\mathbf{A^c}(\overleftarrow{i,j},k)$ *when* $l \neq i,j$ *and*

$\mathbf{A^c}(\overrightarrow{i,j},k)\mathbf{M^c}(l) = \mathbf{M^c}(l)\mathbf{A^c}(\overleftarrow{i,j},-k)$ *when* $l = i$ *or* $l = j$
$\mathbf{A^c}(\overleftarrow{i,j},k)\mathbf{M^c}(l) = \mathbf{M^c}(l)\mathbf{A^c}(\overrightarrow{i,j},-k)$ *when* $l = i$ *or* $l = j$

Lemma 12 (S - A commutation, swap-add commutation)
$\mathbf{S}\ \ \mathbf{A} = \mathbf{A}\ \ \mathbf{S}$ *when they are disjoint and*

$$\mathbf{S^c}(i,j)\mathbf{A^c}(\overrightarrow{i,j},k) = \mathbf{A^c}(\overleftarrow{i,j},k)\mathbf{S^c}(i,j)$$
$$\mathbf{S^c}(i,j)\mathbf{A^c}(\overleftarrow{i,j},k) = \mathbf{A^c}(\overrightarrow{i,j},k)\mathbf{S^c}(i,j)$$

$$\mathbf{S^c}(i,j)\mathbf{A^c}(\overrightarrow{j,l},k) = \mathbf{A^c}(\overrightarrow{i,l},k)\mathbf{S^c}(i,j)$$
$$\mathbf{S^c}(i,j)\mathbf{A^c}(\overleftarrow{j,l},k) = \mathbf{A^c}(\overleftarrow{i,l},k)\mathbf{S^c}(i,j)$$

$$\mathbf{S^c}(j,l)\mathbf{A^c}(\overrightarrow{i,j},k) = \mathbf{A^c}(\overrightarrow{i,l},k)\mathbf{S^c}(j,l)$$
$$\mathbf{S^c}(j,l)\mathbf{A^c}(\overleftarrow{i,j},k) = \mathbf{A^c}(\overleftarrow{i,l},k)\mathbf{S^c}(j,l)$$

$$\mathbf{S^c}(i,l)\mathbf{A^c}(\overrightarrow{j,l},k) = \mathbf{A^c}(\overleftarrow{i,j},k)\mathbf{S^c}(i,l)$$
$$\mathbf{S^c}(i,l)\mathbf{A^c}(\overleftarrow{j,l},k) = \mathbf{A^c}(\overrightarrow{i,j},k)\mathbf{S^c}(i,l)$$

$$\mathbf{S^c}(i,l)\mathbf{A^c}(\overrightarrow{i,j},k) = \mathbf{A^c}(\overleftarrow{j,l},k)\mathbf{S^c}(i,l)$$
$$\mathbf{S^c}(i,l)\mathbf{A^c}(\overleftarrow{i,j},k) = \mathbf{A^c}(\overrightarrow{j,l},k)\mathbf{S^c}(i,l)$$

4.2 Reformulation without direction

Lemmas 10 11 12 can be reformulated with different notation which does not use the directional information. When we write $\mathbf{O}(i \leftrightarrow j)$ resp. $\mathbf{O}(j \oplus k \otimes i)$ we now suppose only $1 \le i$ and $1 \le j$.

Lemma 13 (S - M commutation, swap-multiply commutation)

$\mathbf{SM} = \mathbf{MS}$ *when they are disjoint and*

$\mathbf{O}(i \leftrightarrow j)\mathbf{O}(k) = \mathbf{O}(j)\mathbf{O}(i \leftrightarrow j)$ *when* $k = i$
$\mathbf{O}(i \leftrightarrow j)\mathbf{O}(k) = \mathbf{O}(i)\mathbf{O}(i \leftrightarrow j)$ *when* $k = j$

Lemma 14 (A - M commutation, add-multiply commutation)
$\mathbf{AM} = \mathbf{MA}$ *when they are disjoint and*

$\mathbf{O}(j \oplus k \otimes i)\mathbf{O}(l) = \mathbf{O}(l)\mathbf{O}(i \oplus -k \otimes j)$ *when* $l = i$ *or* $l = j$
$\mathbf{O}(i \oplus k \otimes j)\mathbf{O}(l) = \mathbf{O}(l)\mathbf{O}(j \oplus -k \otimes i)$ *when* $l = i$ *or* $l = j$

Lemma 15 (S - A commutation, swap-add commutation)
$\mathbf{SA} = \mathbf{AS}$ *when they are disjoint and*

$\mathbf{O}(i \leftrightarrow j)\mathbf{O}(j \oplus k \otimes i) = \mathbf{O}(i \oplus k \otimes j)\mathbf{O}(i \leftrightarrow j)$
$\mathbf{O}(i \leftrightarrow j)\mathbf{O}(l \oplus k \otimes j) = \mathbf{O}(l \oplus k \otimes i)\mathbf{O}(i \leftrightarrow j)$
$\mathbf{O}(i \leftrightarrow j)\mathbf{O}(j \oplus k \otimes l) = \mathbf{O}(i \oplus k \otimes l)\mathbf{O}(i \leftrightarrow j)$
$\mathbf{O}(j \leftrightarrow l)\mathbf{O}(j \oplus k \otimes i) = \mathbf{O}(l \oplus k \otimes i)\mathbf{O}(j \leftrightarrow l)$
$\mathbf{O}(j \leftrightarrow l)\mathbf{O}(i \oplus k \otimes j) = \mathbf{O}(i \oplus k \otimes l)\mathbf{O}(j \leftrightarrow l)$

4.3 G_M subgroup

The group of unimodular operations resp. unimodular matrices is also called *general linear group GL(n)*. The following lemma allows us to focus on subgroup of operations of the kind $\mathbf{O}(j \oplus k \otimes i)$ i.e. the add operations.

Lemma 16 (decomposition lemma) *Every unimodular operation can be uniquely decomposed in the following way*

$$\mathbf{U} = \mathbf{SMA}$$

where $\mathbf{S}$ *is an element of a permutation subgroup,* $\mathbf{M}$ *is an element of a subgroup generated by the operations* $\mathbf{O}(i)$ $(1 \leq i \leq n)$ *and* $\mathbf{A}$ *is an element of a subgroup generated by the operations* $\mathbf{O}(j \oplus k \otimes i)$ *.*

The previous decomposition lemma moves the scope of the commutativity investigations into the subgroups G_A, G_S, G_M where the most interesting subgroup for us is G_A. By similar reasoning, as in the above section, it can be shown that

Lemma 17 (A - A commutation, add-add commutation)
$\mathbf{O}(j_1 \oplus k_1 \otimes i_1)$ $\mathbf{O}(j_2 \oplus k_2 \otimes i_2)$ $=$ $\mathbf{O}(j_2 \oplus k_2 \otimes i_2)$ $\mathbf{O}(j_1 \oplus k_1 \otimes i_1)$ *when they are disjoint or in one of the following cases:*

1. $i_1 \neq i_2$ *and* $j_1 = j_2$

2. $i_1 = i_2$ *and* $j_1 = j_2$

3. $i_1 = i_2$ *and* $j_1 \neq j_2$

The following three cases are generally noncomutative:

1. $i_1 \neq j_2$ *and* $j_1 = i_2$

2. $i_1 = j_2$ *and* $j_1 = i_2$

3. $i_1 = j_2$ *and* $j_1 \neq i_2$

References

[1] C. Hermite, *Sur l'Introduction des Variables Continues dans la Theorie des Nombres*, J. Reine Angew. Math. 41, 1851, pp. 191-216.

[2] H. J. S. Smith, *On Systems of Indeterminate Equations and Congruences*, Philos. Trans., 151, 1861, pp. 293-326.

[3] G. H. Hardy and E. M. Wright, *An Introduction to the Theory of Numbers*, The fifth edition, Oxford, 1979 (the first edition, Oxford, 1938).

[4] J. B. Rosser, *A Method of Computing Exact Inverse of Matrices with Integer Coefficients*, J. Res. Nat. Bur. Standarts, 49, 1952, pp. 349-358.

[5] D. A. Smith, *A Basis Algorithm for Finitely Generated Abelian Groups*, Math. Algorithms, 1, 1966, pp. 13-26.

[6] J. Edmonds, *Systems of Distinct Representatives and Linear Algebra*, J. Res. Nat. Bur. Standarts, 71B, 1967, pp. 241-245.

[7] G. Birkhoff and S. MacLane, *Algebra*, MacMillan, London, 1968.

[8] T. C. Hu, *Integer Programming and Network Flows*, Addison Wesley, Reading, Mass., 1969.

[9] G. H. Bradley, *Algorithm and Bound for the Greatest Common Divisor of n Integers*, Comm. ACM, 13, 1970, pp. 433-436.

[10] M. Heymann and J. A. Thorpe, *Transfer Equivalence of Linear Dynamical Systems*, SIAM J. Control Optimization, 8, 1970, pp. 19-40.

[11] G. H. Bradley, *Algorithms for Hermite and Smith Normal Form Matrices and Linear Diophantine Equations*, Math. Comp. 25, 1971, pp. 897-907.

[12] Donald E. Knuth, *The Art of Computer Programming. Vol. II: Seminumerical Algorithms*, Addison Wesley, Reading, Mass., 1971.

[13] M. Newman, *Integral Matrices*, AP New York, 1972.

[14] Donald E. Knuth, *The Art of Computer Programming. Vol. III: Sorting and Searching*, Addison Wesley, Reading, Mass., 1973.

[15] G. A. Gorry, W. D. Northup and J. F. Shapiro, *Computational Experience with a Group Theoretic Integer Programming Algorithm*, Math. Programming, 4, 1973, pp. 171-192.

[16] M. T. McClellan, *The Exact Solution of Systems of Linear Equations with Polynomial Coefficients*, J. ACM 20, 1973, pp. 563-588.

[17] S. Barnette and I. S. Pace, *Efficient Algorithms for Linear System Calculations; Part I - Smith Form and Common Divisor of Polynomial Matrices*, Internat. J. of Systems Sci., 5, 1974, pp. 403-411.

[18] M. A. Frumkin, *Polynomial Time Algorithms in the Theory of Linear Diophantine Equations*, in M. Karpinski (Ed.), Fundamentals of Computation Theory. Springer, Berlin and New York, 1977, Lecture Notes in Computer Sci. 56, pp. 386-392.

[19] G. Havas and L. Sterling, *Integer Matrices and Abelian Groups. Symbolic and Algebraic Computation*, Lecture Notes in Computer Sci., EUROSAM'79, An International Symposium on Symbolic and Algebraic Manipulation, Marseille, France, 1979.

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

[21] Tsu-Wu J. Chou and G. E. Collins, *Algorithms for the Solution of Systems of Linear Diophantine Equations*, SIAM. J. Comput., Vol. 11, No. 4, November 1982, pp. 687-708.

[22] J. von zur Gathen, *Parallel Algorithms for Algebraic Problems*, SIAM J. Comput., Vol. 13, No. 4, 1984, pp. 802-824.

[23] Vangalur S. Alagar, Asim Kumar Roy, *A Comparative Study of Algorithms for Computing the Smith Normal Form of an Integer Matrix*, Int. J. Systems Sci., Vol. 15, No. 7, 1984, pp. 727-744.

[24] Stephen A. Cook, *A Taxonomy of Problems with Fast Parallel Algorithms*, Information and Control, Vol. 64, 1985, pp. 2-22.

[25] P. D. Domich, R. Kannan and L. E. Trotter, Jr., *Hermite Normal form Computation Using Modulo Determinant Arithmetic*, Math. of Operating Research, Vol. 12, No. 1, February 1987, pp. 50-59.

[26] Ravindran Kannan, Garry Miller, Larry Rudolph, *Sublinear Parallel Algorithm for Computing the Greatest Common Divisor of Two Integers*, SIAM J. Comput., Vol. 16, No.1, February 1987, pp. 7-16.

[27] E. Kaltofen, *Computer Algebra Algorithms*, Annual Review of Computer Science, Vol. 2, 1987, pp. 91-118.

[28] Erich Kaltofen, M. S. Krishnamoorthy, B. David Saunders, *Fast Parallel Computation of Hermite and Smith Forms of Polynomial Matrices*, SIAM J. Alg. Disc. Meth., Vol. 8, No. 4, October 1987, pp. 683-690.

[29] Costas S. Iliopoulos, *Worst-Case Complexity Bounds on Algorithms for Computing the Canonical Structure of Finite Abelian Groups and the Hermite and Smith Normal Forms of an Integer Matrix*, SIAM. J. Comput., Vol. 18, No. 4, August 1989, pp. 658-669.

[30] Costas S. Iliopoulos, *Worst-Case Complexity Bounds on Algorithms for Computing the Canonical Structure of Infinite Abelian Groups and Solving Systems of Linear Diophantine Equations*, SIAM. J. Comput., Vol. 18, No. 4, August 1989, pp. 670-678.

[31] Paul D. Domich, *Residual Hermite Normal Form Computations*, ACM Transactions on Mathematical Software, Vol. 15, No. 3, September 1989, pp. 275-286.

[32] F. Siebert Roch, *Solving Linear Diophantine Equations in Parallel*, Computer Algebra and Parallelism, Academic Press, London, 1989, pp. 208-222.

[33] Benny Chor, Oded Goldreich, *An Improved Parallel Algorithm for Integer GCD*, Algorithmica, Vol. 5, 1990, pp. 1-10.

[34] Shing-Tsaan Huang, *Notes on Shuffle/Exchange Type Permutation sets*, IEEE Transactions on Computers, Vol. 39, No. 7., July 1990, pp. 962-965.

[35] Erich Kaltofen, M. S. Krishnamoorthy, B. David Saunders, *Parallel Algorithms for Matrix Normal Forms*, Linear Algebra and its Applications, 136, 1990, pp. 189-208.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 46 avenue Félix Viallet, 38031 GRENOBLE Cedex 1
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399