A Parallel Lattice Basis Reduction for Mesh-connected Processor Arrays and Parallel Complexity

C. Heckler L. Thiele

Universität des Saarlandes, Lehrstuhl für Mikroelektronik
Postfach 1150, 66041 Saarbrücken, Germany

Abstract

Lattice basis reduction has important applications in the areas of computer algebra, cryptography and combinatorial optimization. Several efficient sequential algorithms are known. Recently, parallel algorithms have been developed but until now a formal proof for the efficiency of parallel algorithms with n^2 processors has been omitted, where n denotes the dimension of the lattice. In this paper, a variant of the well-known basis reduction algorithms is presented that is well suited for the computation with fast floating point arithmetic and for the implementation on a mesh-connected array of n^2 processors. In addition, an error analysis and a proof of the parallel efficiency is provided.

1 Introduction

Finding short vectors in a lattice is an important problem in the areas of computer algebra and geometry of numbers. Important applications of basis reduction are for example polynomial factorization [9], knapsack or subset sum problems [13], knapsack public-key cryptosystems [10] and factorization over algebraic number fields [8]. There are several sequential polynomial time algorithms that find a basis of a lattice with short vectors ('reduced basis') for example the well-known LLL algorithm [9] and others [12]. Recently, Roch and Villard [16, 11] developed parallel algorithms for n and n^2 processors, where n denotes the dimension of the lattice. Based on many experimental results, they claim that their parallel implementations are efficient. However, to our knowledge, they have not yet published a corresponding proof for the case of n^2 processors. In addition, their algorithms use slow exact integer arithmetic. In order to avoid the drawbacks of the slow exact integer arithmetic, Schnorr and Euchner [13] have proposed a sequential algorithm which uses fast floating point arithmetic. An efficient parallel implementation of this algorithm for a ring of n processors can easily be derived. But the algorithm cannot be parallelized efficiently for n^2 processors. A new variant of the classical basis reduction algorithms can be derived that is well suited for the computation with fast floating point arithmetic and for the parallelization on a mesh-connected array of n^2 processors, see e.g. [6]. In this paper it is shown that, in comparison with the LLL algorithm, a speed-up of n^2 (with respect to arithmetic operations and communications of numbers) can be achieved to find a short vector and an error analysis is given.

The next section gives a brief survey of the principal concepts for basis reduction is given. In section 3 the new (sequential) algorithm, an error analysis and practical results are presented. The parallel implementation of the algorithm on a mesh-connected array is treated in section 4. Finally, in section 5 the time complexity of the parallel implementation is considered.

2 Foundations of basis reduction

Basic concepts and sequential algorithms for lattice basis reduction are described in [9, 13, 14, 5]. Here, we only give a brief summary.

2.1 Lattices and reduced bases

Let $b_1, b_2, \ldots, b_n \in \mathbf{R}^n$ be n linearly independent vectors. Then

$$L = L(b_1, \ldots, b_n) = \left\{ \sum_{i=1}^{n} \lambda_i b_i | \lambda_i \in \mathbf{Z}, i = 1, \ldots, n \right\}$$

is called the lattice with basis $(b_1, \ldots, b_n)$. The aim of basis reduction is to find short vectors of the lattice or a basis with short vectors.

If $A = (a_{i,j})_{1 \le i,j \le n}$ is a unimodular matrix, i.e. $A \in \mathbf{Z}^{n \times n}$ and $\det A = \pm 1$, then $(b'_1, \ldots, b'_n)$ with $b'_i := \sum_{j=1}^{n} a_{i,j} b_j$ is another basis of the same lattice. All bases of the lattice arise from such transformations. Therefore, $\det L := |\det(b_1, \ldots, b_n)|$ is independent of the basis of the lattice.

Throughout this paper, $(b_1, b_2, \ldots, b_n)$ are the (current) basis vectors of a lattice and $(b_1^*, b_2^*, \ldots, b_n^*)$ the associated Gram-Schmidt orthogonalization, i.e.

$$b_1^* = b_1, \quad b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{i,j} b_j^* \quad \text{for} \quad i = 2, \ldots, n$$

where $\mu_{i,j} = \frac{b_i^T b_j^*}{||b_j^*||^2}$. Let be $\mu_{i,i} = 1$ and $\mu_{i,j} = 0$ for $i < j$ and $M = (\mu_{i,j})_{1 \le i,j \le n}$ the matrix of the Gram-Schmidt coefficients. That is, M is a lower triangular matrix where all elements on the main diagonal are equal to 1. Then we can write the above equations as $(b_1, \ldots, b_n) = (b_1^*, \ldots, b_n^*) M^T$. In the following, μ_j denotes the j-th column of M^T. For any basis of the lattice L, the Gram-Schmidt orthogonalization satisfies $\det L = |\det(b_1^*, b_2^*, \ldots, b_n^*)| = ||b_1^*|| \cdot ||b_2^*|| \cdot \ldots \cdot ||b_n^*||$. In order to find short vectors we use the following definition of a reduced basis (due to [14]) which is almost the same as the one given in [9, 11, 5].

Definition 2.1 (Reduced basis [14]) *A lattice basis $(b_1, \ldots, b_n)$ is called reduced, if the following two conditions hold*

- $|\mu_{i,j}| \le \frac{1}{2}$ *for all* $1 \le j < i \le n$
- $||b_i^*||^2 \le 2||b_{i+1}^*||^2$

Theorem 2.1 ([5, Theorem 5.3.13]) *Let L be a lattice in $\mathbf{R}^n$ and $(b_1, \ldots, b_n)$ be a reduced basis*[1] *of L. Then the following holds: $||b_1|| \le 2^{\frac{n-1}{4}} (\det L)^{\frac{1}{n}}$.*

2.2 Structure of reduction algorithms

The basic idea of the LLL reduction algorithm of [9] can be formulated as follows (see [5, 14]).

Algorithm 2.1

Basic Lattice Basis Reduction Algorithm (due to [5, 14])
input: $b_1, b_2, \ldots, b_n$ a basis
output: $b_1, b_2, \ldots, b_n$ a reduced basis
method:

```
swapping ← true
while(swapping) do
    Compute the GS-orthogonalization:
    ||b_1*||^2, ..., ||b_n*||^2 and M
    (a) for i = 2 to n do
        for j = i − 1 downto 1 do
            Let ⌈μ_{i,j}⌋ be the integer
            nearest to μ_{i,j}
            b_i ← b_i − ⌈μ_{i,j}⌋ b_j
            μ_i ← μ_i − ⌈μ_{i,j}⌋ μ_j
        od
    od
    swapping ← false
    (b) if (there is an index i with
      ||b_i*||^2 > 2||b_{i+1}*||^2) then
        swapping ← true
        For such an index i swap b_i and
        b_{i+1}, i.e. consider the
        new basis (b_1, ..., b_{i+1}, b_i, ..., b_n).
    fi
od
```

After step (a) the first condition of a reduced basis is satisfied ('size reduction of M'). Now, consider the swapping step (b) and let b_j^* and $b_j^{*'}, (1 \le j \le n)$ denote the vectors of the Gram-Schmidt orthogonalization before and after a swapping operation, respectively. Then, it can easily be verified that, after swapping b_i and b_{i+1}, the new vectors satisfy [9, 5]:

$$\begin{aligned} b_j^{*'} &= b_j^* \text{ if } j \in \{1, \ldots, n\} \setminus \{i, i+1\} \\ b_i^{*'} &= b_{i+1}^* + \mu_{i+1,i} \cdot b_i^* \\ ||b_i^{*'}||^2 &\le \tfrac{3}{4} ||b_i^*||^2 \qquad (1) \\ ||b_i^{*'}||^2 \cdot ||b_{i+1}^{*'}||^2 &= ||b_i^*||^2 \cdot ||b_{i+1}^*||^2 \qquad (2) \end{aligned}$$

In order to examine the complexity of algorithm 2.1 the number of swaps which are carried out during the reduction is considered.

Let $B = \max_{i=1,\ldots,n} ||b_i||^2$ be the square of the length of the longest vector of the original basis. This constant B plays a central role in the determination of the time complexity of basis reduction algorithms. The Gram-Schmidt orthogonalization yields $||b_i^*||^2 \le B$ for $i = 1, \ldots, n$.

The following theorem gives an upper bound to the number of swaps which have to be executed during the reduction algorithm:

Theorem 2.2 (Sequential Time [9, 5]) *The algorithm 2.1 terminates after at most $n^2 \log B$ swaps, if we start with an integral basis.*[2]

[1] The theorem is true, although our definition of a reduced basis is slightly different from that given in [5].

[2] In the following, log denotes the logarithm with basis $\frac{4}{3}$.

Proof: Consider the quantity

$$D := ||b_1^*||^{2n} \cdot ||b_2^*||^{2(n-1)} \cdot \ldots \cdot ||b_n^*||^2$$

At the beginning $D \leq B^{(n^2)}$. If we start with an integral basis, D is also an integer and remains integral during the whole algorithm, i.e. $D \geq 1$ (see [5]). Each swap decreases the value of D by a factor less than $\frac{3}{4}$. If l denotes the number of swaps, we have

$$1 \leq \left(\frac{3}{4}\right)^l D \leq B^{(n^2)} \left(\frac{3}{4}\right)^l \Rightarrow l \leq n^2 \log B$$

■

Corollary 2.3
Algorithm 2.1 needs at most $O(n^5 \log B)$ arithmetic operations.

Proof: Each swapping operation induces a recalculation of the Gram-Schmidt orthogonalization and an execution of the size reduction step (a). Both steps need $O(n^3)$ arithmetic operations. At most $n^2 \log B$ swaps are executed. Therefore, the given bound results. ■

The LLL algorithm [9] and other sequential algorithms based on it use a more sophisticated strategy. The main idea is to compute the Gram-Schmidt orthogonalization and its size reduction (step (a) of algorithm 2.1) of only one column per swap operation. Thus, only $O(n^2)$ arithmetic operations are executed per swap. Consequently, the following corollary holds:

Corollary 2.4 *There are sequential basis reduction algorithms that need $O(n^4 \log B)$ arithmetic operations.*

An efficient parallel implementation of the LLL-algorithm and the floating point variant of Schnorr and Euchner [13] on a ring of n processors is possible. Each phase consisting in the computation of a column k of the Gram-Schmidt coefficients, the size-reduction of that column and one swap can be executed with $O(n)$ parallel arithmetic operations. To this end, the matrix $(b_1, \ldots, b_n)$ of the basis vectors and the matrix M^{T} of the Gram-Schmidt orthogonalization are distributed by rows over the processors. Besides, in each processor i the column i of matrix M^{T} is stored. The computation of the inner products for the Gram-Schmidt orthogonalization is parallelized like a matrix-vector multiplication (see [1]). The computation of the coefficients $\mu_{k,i}$ is parallelized with pipelining methods using the distribution by columns. Each processor i with $1 \leq i < k$ computes the value of $\mu_{k,i}$. The size-reduction of the computed column k is computed in parallel using pipelining methods, too. But to this end, the distribution by rows of the coefficients $\mu_{i,j}$ has to be used. If no swap has to be executed, the new computed column k of M^{T} is sent to processor k. Thus, the correct distribution of data is guaranteed. See [11, 16] for the LLL-algorithm.

3 A new variant of LLL reduction

In contrast to the case of n processors, the LLL-algorithm [9] and the floating point variant of Schnorr and Euchner [13] doesn't seem to be parallelizable with n^2 processors because of their 'sequential nature' which consists in the computation in stages. In each stage only one column is computed. Therefore, an algorithm similar to the basic algorithm 2.1 is used which executes simultaneously operations on the whole matrix.

The crucial idea is to execute all possible swaps in one phase.[3] In one phase of the algorithm, b_i and b_{i+1} are swapped for *all odd* indices i with $||b_i^*||^2 > 2||b_{i+1}^*||^2$ and in the next phase for *all even* indices i. The concatenation of an odd and an even phase is called an '*all-swap phase*'. An algorithm based on that strategy is called '*all-swap algorithm*'. The efficiency of this method is shown in section 5. In order to avoid the slowness of exact integer arithmetic, we follow the idea of Schnorr and Euchner [13] of using floating point arithmetic. Only the lattice basis $(b_1, \ldots, b_n)$ is computed in exact representation. The values of $||b_i^*||^2$ and $\mu_{i,j}$ are stored in floating point format. The following measures are taken in order to minimize the rounding errors:

- Unlike in the LLL algorithm, the Gram-Schmidt orthogonalization is computed in each phase of the algorithm. Thus, rounding errors in the matrix M are corrected in each phase.
- The Gram-Schmidt orthogonalization is computed by means of Givens rotations. Unlike the original Gram-Schmidt procedure (see e.g. [3]), the orthogonalization with Givens rotations is parallelizable with n^2 processors and is numerically stable. An analysis of the possible rounding errors is provided in the next section. Furthermore, we derive a criterion to prevent swaps on account of faulty data.

[3] Roch and Villard [16, 11] had the same idea, too, but they use exact arithmetic. Besides, they could not prove the efficiency of this method.

Givens rotations are used in the following way: As seen in section 2 the Gram-Schmidt orthogonalization can be written in matrix form as

$$(b_1, \ldots, b_n) = (b_1^*, \ldots, b_n^*)M^T$$

M^T is an upper triangular matrix where all elements of the main diagonal are equal to 1 and $b_1^*, \ldots, b_n^*$ are pairwise orthogonal. Only M^T and the norms of $b_1^*, \ldots, b_n^*$, i.e. $||b_1^*||^2, \ldots, ||b_n^*||^2$, have to be computed for the basis reduction. We can use Givens rotations (see e.g. [3]) in order to triangulate $(b_1, \ldots b_n)$, i.e. in order to compute an orthogonal matrix Q with $Q(b_1, \ldots b_n) = R$, where R is upper triangular. Therefore, $(b_1, \ldots, b_n) = Q^T R$ holds. Thus, the norms of the vectors of the Gram-Schmidt orthogonalization can be computed by $||b_i^*||^2 = r_{i,i}^2$ and the matrix of the coefficients M^T is computed from R by dividing each row i by $r_{i,i}$.

Remark: The Gram-Schmidt orthogonalization can also exactly be computed with Givens rotations using a division and square root free method [4]. This method can be used for a fast parallel implementation, too.

Summing up, the basic all-swap floating-point algorithm has the following structure (see also [6]):

Algorithm 3.1

All-Swap Floating Point Lattice Basis Reduction Algorithm
input: exact basis $b_1, b_2, \ldots, b_n$
output: $b_1, b_2, \ldots, b_n$ a reduced[4] basis
method:
ordering $\leftarrow$ odd
while (any swap is possible) do
  $MT \leftarrow$ floating-point$(b_1, \ldots, b_n)$
  (MT corresponds to M^{T})
  Compute the GS-orthogonalization by Givens rotations on MT

  Compute the size reduction of the basis $(b_1, \ldots, b_n)$ and the Gram-Schmidt coefficients MT
  (see step (a) of algorithm 2.1)

  if (ordering = odd) then
    swap b_i and b_{i+1} for every odd index i with $||b_i^*||^2 > 2||b_{i+1}^*||^2$
    ordering $\leftarrow$ even
  else
    swap b_i and b_{i+1} for every even index i with $||b_i^*||^2 > 2||b_{i+1}^*||^2$
    ordering $\leftarrow$ odd
  fi
od

[4] Reduced up to floating-point errors. It can happen, that the resulting basis does not exactly satisfy the conditions of a reduced basis.

3.1 Error analysis

As seen above, the norms $||b_i^*||^2$ of the Gram-Schmidt orthogonalization are computed with floating point arithmetic. Thus, the condition for a swap $||b_i^*||^2 > 2||b_{i+1}^*||^2$ cannot be evaluated in an exact way. In order to avoid incorrect swaps, we derive a condition for the correctness of a swap. To that end, an error analysis of the resulting floating point errors is given. First the possible floating point errors in the Gram-Schmidt orthogonalization are estimated. Then, a condition is derived, whether these floating point errors are too big.

Rounding errors by the computation of the orthogonalization: Here we use an error analysis given by Wilkinson in [17]. Errors are assumed to arise from rounding the (correct) result of an operation to the nearest floating point number. This rounding operation is denoted by $fl(.)$.

Let x be a real number. Then

$$fl(x) = x(1 + \epsilon)$$

with $|\epsilon| \leq u$ and $u = \frac{1}{2}b^{-r+1}$, where b denotes the basis and r the precision of the floating point representation.

Now, we investigate the rounding errors involved in one Givens rotation. Let a be an (exact) vector and G be an (exact) Givens rotation matrix. Disturbed values are denoted by $'$. Let $b = Ga$ be the exactly transformed vector. Then, the disturbed vector $b' = fl(G'a)$ is computed. An analysis due to Wilkinson [17] yields

$$||b' - b|| = ||fl(G'a) - Ga|| \leq 7u||a||$$

A sequence of such Givens rotations is used in order to orthogonalize a matrix A. Let a_i be a column of matrix A and b_i be a column of the transformed matrix. Due to an analysis of Gentleman [2] the error $||b_i' - b_i||$ satisfies

$$||b_i' - b_i|| \leq 7u(2n - 3)(1 + 7u)^{2n-4}||a_i||$$

where n denotes the dimension of the lattice.

For $7u \ll 1$ the term $(1+7u)^{2n}$ can be estimated by $1 + 14 \cdot u \cdot n$ and therefore for $n \cdot u \ll 1$:

$$\begin{aligned} ||b_i' - b_i|| &\leq 7u(2n-3)(1+7u)^{2n-4}||a_i|| \\ &\leq 7u(2n-3)(1+7u)^{2n}||a_i|| \\ &\approx 7 \cdot u(2n-3)(1+14 \cdot u \cdot n)||a_i|| \\ &\leq (14 \cdot u \cdot n + (14 \cdot u \cdot n)^2)||a_i|| \\ &\approx 14 \cdot u \cdot n||a_i|| \end{aligned}$$

This estimation is particularly valid for a single element of the vectors b_i, i.e. $|b_{i,j}' - b_{i,j}| \leq 14 \cdot u \cdot n||a_i||$ for $j = 1, \ldots, n$.

Avoiding wrong swaps: Now, the possible floating-point errors in the condition for a swap $||b_i^{*'}||^2 > 2||b_{i+1}^{*'}||^2$ are computed.

Let $(b_1, \ldots, b_n)$ be the basis of the lattice and $R = Q(b_1, \ldots, b_n)$ the transformed matrix. As seen above, the norms $||b_i^*||$ hold $||b_i^*||^2 = r_{ii}^2 \leq B$.

With the foregoing error analysis the rounding errors in the elements $r_{i,j}$ of the transformed matrix R can be estimated as follows:

$$|r_{i,j}' - r_{i,j}| \leq 14 \cdot n \cdot u \cdot ||b_j||$$

or

$$r_{i,j}' = r_{i,j} + \epsilon$$

with $|\epsilon| \leq 14 \cdot n \cdot u \cdot ||b_j||$.

Two basis vectors b_j and b_{j+1} should be swapped, if for the exact values the relation

$$||b_{j+1}^*||^2 < \frac{1}{2}||b_j^*||^2$$

holds. The factor $\frac{1}{2}$ is arbitrary. A swap improves the basis if the condition is valid for any factor between 0 and $\frac{3}{4}$ (see equation (1)).

If the condition is valid for the disturbed values, the exact values satisfy the inequality

$$(r_{j+1,j+1} + \epsilon_1)^2(1+\epsilon_2) < \frac{1}{2}(r_{j,j} + \epsilon_3)^2(1+\epsilon_4)$$

where

$$\begin{aligned} |\epsilon_1| &\leq 14 \cdot n \cdot u \cdot ||b_{j+1}|| & |\epsilon_2| &\leq u \\ |\epsilon_3| &\leq 14 \cdot n \cdot u \cdot ||b_j|| & |\epsilon_4| &\leq 2u \end{aligned}$$

In addition to the errors in $r_{j+1,j+1}$ and $r_{j,j}$, further errors arise from executing the operations 'square' and 'dividing by two'. These errors are estimated by ϵ_2 and ϵ_4.

After some computations and neglecting terms of higher degree and terms in the order of magnitude of $u \cdot r_{j,j}$ and $u \cdot r_{j+1,j+1}$ the following inequality holds:

$$r_{j+1,j+1}^2 + (2\epsilon_1 r_{j+1,j+1} + \epsilon_1^2 - r_{j,j}\epsilon_3 - \frac{1}{2}\epsilon_3^2) < \frac{1}{2}r_{j,j}^2$$

for $u \ll 1$.

The error is estimated by

$$|2\epsilon_1 r_{j+1,j+1} + \epsilon_1^2 - r_{j,j}\epsilon_3 - \frac{1}{2}\epsilon_3^2| \leq 28nu\sqrt{nB}|r_{j+1,j+1}| +$$

$$14^2 n^3 u^2 B + 14nu\sqrt{nB}|r_{j,j}| + \frac{1}{2}(14^2 n^3 u^2 B)$$

($||b_i||^2 \leq nB$ holds for all i during the execution of the whole algorithm [9].)

If the following inequality is valid

$$28nu\sqrt{nB}|r_{j+1,j+1}| + 14nu\sqrt{nB}|r_{j,j}| + \frac{3}{2}14^2 n^3 B u^2$$

$$< \frac{1}{4}r_{j,j}^2$$

the exact values satisfy $r_{j+1,j+1}^2 < \frac{3}{4}r_{j,j}^2$ and henceforth, a swap should be executed.

The following theorem summarizes the preceding discussion:

Theorem 3.1 *Using the floating-point error model of Wilkinson [17], a swap will result in a better basis, if the condition $||b_j^{*'}||^2 > 2||b_{j+1}^{*'}||^2$ is valid for the disturbed values and*

$$n^3 B u^2 \ll r_{j,j}^2$$

for $u \ll 1$, $n \in \mathbf{N}$ and $n \cdot u \ll 1$.

Proof: See discussion above. ■

If no swap is possible because of the rounding errors, one exact Gram-Schmidt computation has to be executed.

Remark: In each phase, the value of B in the swap condition can be replaced by $\max_{i=1,\ldots,n} ||b_i||^2$ of the actual lattice basis.

3.2 Practical results

The algorithm was implemented in C++ using standard classes for exact arithmetic. The implementation was not optimized regarding the computing time. It has been tested on a Sun Sparcstation 10 for two kinds of lattices: lattices which are generated from the identity matrix by iteratively multiplying with random unimodular 2×2 matrices (see

n	B	na	ns	s/a	time
5	$1.98 \cdot 10^{45}$	34	101	3	0.5
10	$5.18 \cdot 10^{64}$	104	786	7.6	15.56
20	$4.1 \cdot 10^{62}$	239	3765	15.8	303
30	$2.8 \cdot 10^{75}$	518	12012	23.2	2202
40	$2.3 \cdot 10^{82}$	800	23422	29.3	7324
50	$2.11 \cdot 10^{92}$	1018	36844	36.2	16870
60	$2 \cdot 10^{102}$	1519	66190	43.6	41209
70	$2 \cdot 10^{132}$	2501	124439	49.8	112124
80	$7 \cdot 10^{154}$	3609	199917	55.4	240625
90	$4 \cdot 10^{172}$	4988	311243	62.4	507627

Table 1: Results for unimodular transformed lattices

B	na	ns	s/a	time [sec]
$3.09 \cdot 10^{23}$	31	249	8.03	13.4
$1.14 \cdot 10^{44}$	46	503	10.9	28.7
$1.43 \cdot 10^{64}$	65	748	11.5	46.5
$1.65 \cdot 10^{84}$	80	991	12.4	60.6
$4.45 \cdot 10^{103}$	88	1127	12.8	69.4
$1.02 \cdot 10^{124}$	113	1405	12.4	85.5
$1.5 \cdot 10^{144}$	116	1530	13	101.8
$9.62 \cdot 10^{163}$	140	1818	13	121.7
$2.42 \cdot 10^{184}$	148	1988	13.4	131.6
$5.2 \cdot 10^{203}$	163	2185	13.4	144

Table 2: Results for subset sum lattices with $n = 21$

Table 1 and [6]) and subset sum lattices which are constructed due to [13] (see Table 2).

The columns of the tables are labeled as follows:

n	dimension of the lattice
B	$max_{i=1,\ldots,n}\|\|b_i\|\|^2$
na	number of all-swap phases
ns	number of swaps
s/a	average number of swaps per phase
time	CPU time in seconds

The tests show two facts. First, for 'hard' lattices (the unimodular transformed lattices) many swaps can be executed in parallel, and second, the algorithm is well suited for practical problems such as subset sum lattices.

4 Parallel algorithm

One phase of algorithm 3.1 consists in two main steps: the orthogonalization of MT using Givens rotations and the size-reduction. Both steps are parallelizable on a mesh-connected processor, see e.g. Schwiegelshohn and Thiele [15].

They consider algorithms of the form

Algorithm 4.1

for $i = 1$ to $n-1$ do
  for $j = i+1$ to n do
    $A \leftarrow f_{ij}(A)$

where f_{ij} is a function of the following form

$$a_{ik} \leftarrow p_1 a_{ik} + p_2 a_{jk} \qquad a_{jk} \leftarrow p_3 a_{ik} + p_4 a_{jk}$$
$$a_{ki} \leftarrow p_5 a_{ki} + p_6 a_{kj} \qquad a_{kj} \leftarrow p_7 a_{ki} + p_8 a_{kj}$$

for $k = 1, \ldots, n$, where the parameters p_l ($l = 1, \ldots, 8$) are computed from the matrix elements $a_{ii}, a_{ij}, a_{ji}, a_{jj}$.

They give a realization on a mesh-connected processor array for the following parallel version:

Algorithm 4.2

for $k = 3$ to $2n-1$ do
  for $i = \max\{1, k-n\}$ to $(k-1)/2$ pardo
    $j = k - i$
    $A \leftarrow f_{ij}(A)$

Since the orthogonalization using Givens rotations and the size reduction of M obey the restrictions with respect to the functions f_{ij}, algorithm 3.1 can easily be parallelized in this way (see also [6]). Thus, the follwoing theorem holds (Villard [16] shows a similar result for the parallel time of one phase using exact arithmetic and other methods for parallelization).

Theorem 4.1 *One phase of the all swap basis reduction algorithm can be executed in $O(n)$ parallel steps (considering the number of arithmetic operations and the number of communications of numbers) on a mesh-connected processor array with $O(n^2)$ processors cells.*

5 Time complexity

In order to prove the efficiency of the new algorithm, we give a theorem on the number of phases that have to be executed to find a short vector using all-swap algorithms. As described in section 4, an all-swap phase including the computation of the

Gram-Schmidt orthogonalizations and the size reductions can be done in $O(n)$ parallel arithmetic operations with n^2 processors[5]. Therefore, a complexity of $O(l \cdot n)$ arithmetic operations is obtained, where l denotes the number of necessary all-swap phases. In order to achieve a speed-up of n^2 in comparison with the LLL algorithm, only $n \log B$ all-swap phases should be executed (see corollary 2.4).

Theorem 5.1 (Parallel Time Complexity) *After at most $n \log B$ all-swap phases the following situation holds for all i with $1 \leq i \leq n$*

$$||b_1^*||^2 \cdot ||b_2^*||^2 \cdot \ldots \cdot ||b_i^*||^2 \leq \tfrac{4}{3} \cdot c^{\frac{i(n-i)}{2}} \cdot (\det L)^{\frac{2i}{n}}$$

with $c = \frac{32}{9}$.

Proof: Consider the rates

$$\frac{||b_1^*||^2}{c^{\frac{n-1}{2}}(\det L)^{\frac{2}{n}}} \cdot \frac{||b_2^*||^2}{c^{\frac{n-3}{2}}(\det L)^{\frac{2}{n}}} \cdot \ldots \cdot \frac{||b_i^*||^2}{c^{\frac{n-2i+1}{2}}(\det L)^{\frac{2}{n}}}$$

$$= \frac{\prod_{j=1}^{i} ||b_j^*||^2}{c^{\frac{i(n-i)}{2}}(\det L)^{\frac{2i}{n}}} =: v(i) \qquad (3)$$

We are finished when all these rates $v(i), 1 \leq i \leq n$, are less than $\frac{4}{3}$. Let max be the maximum of all these rates, i.e. $max = \max\{v(i) | 1 \leq i \leq n\}$ and max' the maximum after one further all-swap phase.

It will be shown that every all-swap phase decreases the value of max by a factor less than $\frac{3}{4}$, i.e. $max' \leq \frac{3}{4} max$, if $max > \frac{4}{3}$. At the beginning, $max \leq B^n$ since $||b_i^*||^2 \leq B$ holds for all $1 \leq i \leq n$. Therefore, all rates are less than $\frac{4}{3}$ after at most $n \log B$ all-swap phases.

Proposition 5.1 *If $max > \frac{4}{3}$ and $v(i) > \frac{3}{4} max$, then the product $||b_1^*||^2 \cdot ||b_2^*||^2 \cdot \ldots \cdot ||b_i^*||^2$ and thereby $v(i)$ is decreased by a factor less than $\frac{3}{4}$ in one all-swap phase. All other rates remain unchanged or decrease.*

Proof of the proposition:

If two vectors b_j and b_{j+1} are swapped, then the product $||b_1^*||^2 \cdot ||b_2^*||^2 \cdot \ldots \cdot ||b_j^*||^2$ is decreased by a factor less than $\frac{3}{4}$, see (1). All other products remain unchanged, see (2). Thus, we have to show that $max \leq \frac{4}{3}$ or b_i and b_{i+1} are swapped for all i with $v(i) > \frac{3}{4} max$ in one all-swap phase, i.e. if $v(i) > \frac{3}{4} max$, then $||b_i^*||^2 > 2||b_{i+1}^*||^2$. If this swap condition holds for an even index i, then it also holds after one odd swapping phase (see (1) and (2)).

If $v(n) > \frac{3}{4} max$, then $max < \frac{4}{3}$ since $v(n) = 1$.

Therefore, consider an index $i, 1 \leq i \leq n-1$, with $v(i) > \frac{3}{4} max$. We show in the following that the vectors b_i and b_{i+1} are swapped in this case.

If $v(i) > \frac{3}{4} max$, then $v(i) > \frac{3}{4} v(i+1)$ and using (3) we obtain:

$$\frac{\frac{4}{3}(\det L)^{\frac{2}{n}} c^{\frac{n-2i-1}{2}}}{||b_i^*||^2} > \frac{||b_{i+1}^*||^2}{||b_i^*||^2} \qquad (4)$$

In the same way, if $v(i) > \frac{3}{4} max$ and $i > 1$, then $v(i) > \frac{3}{4} v(i-1)$ holds and therefore

$$||b_i^*||^2 > \frac{3}{4} c^{\frac{n-2i+1}{2}} (\det L)^{\frac{2}{n}} \qquad (5)$$

This relation holds also for $i = 1$, if $max \geq 1$ and $v(1) > \frac{3}{4} max$.

However, it follows from (4) and (5) that

$$\frac{\left(\frac{4}{3}\right)^2}{c} = \frac{1}{2} > \frac{||b_{i+1}^*||^2}{||b_i^*||^2} \qquad (6)$$

and therefore the swap condition is satisfied.

Thus, the proposition and the theorem have been proven. ■

In particular, we have found a short vector as shown in the following corollary:

Corollary 5.2 *After at most $n \log B$ all-swap phases the vector b_1 of the resulting lattice basis satisfies*

$$||b_1|| \leq \tfrac{4}{3} c^{\frac{n-1}{4}} (\det L)^{\frac{1}{n}}$$

Proof: Use theorem 5.1 with $i = 1$ and $b_1 = b_1^*$. ■

This is almost the same bound as in theorem 2.1.

Corollary 5.3 *A short vector of a lattice can be found in $O(n^2 \log B)$ parallel steps using a network of n^2 processors.*

Proof: See corollary 5.2 and theorem 4.1. ■

Until now, we have considered the complexity in terms of arithmetic operations. If we want to consider the binary complexity, we have to compute the time for one exact arithmetic operation. This time depends on the size of the numbers. Using all-swap algorithms, bigger numbers can arise than in the original LLL algorithm. Besides, in order to execute the size reduction in parallel, columns are computed with non-reduced columns. Thus, the numbers can grow

[5] This can be done with inexact arithmetic (Givens rotations) or exactly (with exact Givens rotations or methods described in [16]). For the sake of the analysis of the complexity of all-swap algorithms, exact arithmetic is assumed. Using floating point arithmetic, incorrect swaps might be executed or correct swaps might not.

more than in the case of a seriell reduction [16]. However *practical results* of [16] and section 3.2 show that this phenomenon cannot be observed. Besides, there is a special class of interesting lattices ("subset sum lattices") for which the occuring numbers are not that big [16]. Moreover, we have shown that floating point arithmetic can be used. In this case, fewer exact operations have to be executed. Therefore, the size of the numbers is not so weighty. From a *theoretical point of view* a good result can be achieved by parallelizing the exact arithmetic operations. In this case, the size of the numbers is not so important for the time complexity (see also [16]). The crucial result for the parallel efficiency is the new Corollary 5.2.

6 Conclusion

We have shown, that a parallel implementation of lattice basis reduction on a mesh-connected array with n^2 processors is possible. A short vector of a lattice can be found in $O(n^2 \log B)$ parallel steps (considering the number of arithmetic operations and the number of communications of numbers). Thus, a speedup of n^2 in comparison with the sequential case can be achieved. This implementation is efficient in practice, because fast floating point arithmetic can be used. An error analysis was provided in order to avoid wrong swaps. However, from a theoretical point of view, it is still an open problem, why big numbers during the computation do not arise. A good theoretical result can be achieved by parallelizing the exact arithmetic operations.

Note: Recently, Joux has found a method for avoiding big numbers [7].

References

[1] D.P. Bertsekas and J.N. Tsitsiklis. *Parallel and Distributed Computation - Numerical Methods.* Prentice Hall International Editions, 1989.

[2] W. M. Gentleman. Error analysis of QR decompositions by Givens transformations. *Linear Algebra and Its Applications*, 10:189–197, 1975.

[3] G.H. Golub and C.F. van Loan. *Matrix Computations.* The Johns Hopkins University Press, Baltimore, Maryland, 1983.

[4] J. Götze and U. Schwiegelshohn. A square root and division free Givens rotation for solving least squares problems on systolic arrays. *SIAM J. Sci. Stat. Comput.*, 12(4):800–807, 1991.

[5] M. Grötschel, L. Lovasz, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization.* Springer-Verlag, 1988.

[6] C. Heckler and L. Thiele. Parallel complexity of lattice basis reduction and a floating-point parallel algorithm. In *Proc PARLE'93 (Munich), LNCS 694*, pages 744–747. Springer-Verlag, 1993.

[7] A. Joux. A fast parallel lattice reduction algorithm. In *Proc Second Gauss Symposium (Munich)*, 1993.

[8] A.K. Lenstra. Factoring polynomials over algebraic number fields. In *Proc EUROCAL83 (London), LNCS 162*, pages 245–254. Springer-Verlag, 1983.

[9] A.K. Lenstra, H.W. Lenstra Jr., and L. Lovasz. Factoring polynomials with rational coefficients. *Math. Ann.*, 261:515–534, 1982.

[10] R.C. Merkle and M.E. Hellmann. Hiding information and signatures in trap-door knapsacks. *IEEE Trans. Inf. Theory*, 24:525–530, 1978.

[11] J.L. Roch and G. Villard. Parallel gcd and lattice basis reduction. In *Proc CONPAR92 (Lyon), LNCS 632*, pages 557–564. Springer-Verlag, 1992.

[12] C.P. Schnorr. A more efficient algorithm for lattice basis reduction. *J. Algorithms*, 9:47–62, 1988.

[13] C.P. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. In *Proceedings of the FCT'91 (Gosen, Germany), LNCS 529*, pages 68–85. Springer, 1991.

[14] A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley and Sons, 1986.

[15] U. Schwiegelshohn and L. Thiele. A systolic array for cyclic-by-rows Jacobi algorithms. *Journal of Parallel and Distributed Computing*, 4:334–340, 1987.

[16] G. Villard. Parallel lattice basis reduction. In *International Symposium on Symbolic and Algebraic Computation, Berkeley California USA*, pages 269–277. ACM Press, 1992.

[17] J. H. Wilkinson. *The Algebraic Eigenvalue Problem.* Clarendon Press, Oxford, 1965.