Parallel lattice basis reduction

Gilles Villard *

Laboratoire LMC-IMAG, 46 Avenue Félix Viallet
F38031 Grenoble Cédex
e.m. *gvillard@imag.fr*

Résumé

Nous étudions ici la parallélisation de l'algorithme L^3 pour la réduction des bases de réseaux. Sous le modèle des architectures parallèles à mémoires distribuées, l'algorithme proposé permet d'utiliser efficacement $O(n^2)$ processeurs, où n est la dimension de la base considérée. Cet algorithme, implanté sur une machine massivement parallèle, donne lieu à de nombreuses expérimentations. Les premières, reportées ici, font d'une part apparaître de bons résultats quant aux accélérations obtenues en pratique pour des grands nombre de processeurs. Elles permettent d'autre part de compléter les connaissances expérimentales sur la complexité séquentielle de L^3 .

Abstract

The famous L^3 algorithm for lattice basis reduction is parallelized. Using a distributed memory architecture computational model, the algorithm we propose efficiently uses $O(n^2)$ processors, where n is the dimension of the basis to reduce. Its implementation, realized on a massively parallel machine, allows us to conduct many experimentations. The first results are presented in this paper. We show that high speed-ups are obtained even for large amounts of processors, and give new empirical knowledges of the L^3 sequential complexity.

*This work was supported in part by the *PRC Mathématiques et Informatique* and by the *Groupement* C^3 of the french *Centre National de la Recherche Scientifique.*

ISSAC '92-7/92/CA, USA
© 1992 ACM 0-89791-490-2/92/0007/0269...$1.50

1 Introduction and Survey

Throughout the paper n and m are positive integers and $b_1, b_2, \ldots, b_n \in \boldsymbol{Z}^m$ n linearly independent vectors (for the general case, see [7, 23]). Without loss of generality we will take $m = n$. The lattice L in $\boldsymbol{Z}^n$ generated by the basis $(b_1, b_2, \ldots, b_n)$ is the additive subgroup $\sum_{i=1}^{n} \boldsymbol{Z} b_i$. We define the lengh B of the basis to be the largest Euclidean norm $|b_k|$, $|b_i| \leq |b_k| = B, for\ 1 \leq i \leq n$. L will also denote the $n \times n$ matrix which rows are the vectors b_i.

Algorithm 1 : L^3 reduction

Input data : $b_1, b_2, \ldots, b_n$ a basis
Compute the $b_1^*, b_2^*, \ldots, b_n^*$ and the μ_{ij}
i:=2
while i < n
  for j = i-1 **to** 1
    if $|\mu_{ij}| > \frac{1}{2}$ **then** $b_i := b_i - \lceil\ \mu_{ij}\ \rfloor b_j$
  if $|b_i^* + \mu_{i,i-1} b_{i-1}^*|^2 t^2 < |b_{i-1}^*|^2$
    then swap b_i and b_{i-1}, i:=i-1 (i≠1)
    else i:=i+1
Output data : $b_1, b_2, \ldots, b_n$ a t-reduced basis.

Sequential point of view. We recall that the reduction *algorithm 1* described in A.Lenstra, H.Lenstra and L.Lovász [18] modifies an input basis $b_1, b_2, \ldots, b_n$ so it satisfies the following t-reduction properties :

$$|\mu_{ij}| \leq \frac{1}{2} \quad for \quad 1 \leq j < i \leq n, \tag{1}$$

and,

$$|b_i^* + \mu_{i,i-1} b_{i-1}^*|^2 t^2 \geq |b_{i-1}^*|^2 \quad for \quad 1 < i \leq n, \tag{2}$$

where the vectors $b_1^*, b_2^*, \ldots, b_n^*$ denote the Gram-Schmidt orthogonalisation of the output basis :

$$b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{ij} b_j^*, \quad 1 \leq i \leq n, \tag{3}$$

$$\mu_{ij} = \frac{(b_i, b_j^*)}{|b_j^*|^2}, \quad 1 \leq j < i \leq n. \tag{4}$$

For the complexity studies, the parameter t in (2) will have its usual value $2/\sqrt{3}$ but could vary from $1+\epsilon$ ($\epsilon > 0$) to $\sqrt{2}$. The lattice basis reduction algorithm has many application areas, such as factoring polynomials [1, 11, 17, 18] or integers [32], breaking cryptosystem [2, 6, 37, 39, 21, 40], diophantine approximation [14], finding integer relations [7, 8] or solving subset sum problems [15, 24, 33]. Such a great interest for the L^3 fundamental technique has leaded several authors to improve the original method and reduce its computational cost. Assuming that the classical arithmetic procedures are employed, the number of arithmetic operations needed in [18] for the reduction is at most $O(n^4 \log B)$ and the integers on which these operations are performed each have binary length $O(n \log B)$. The algorithm runs in $O(n^6 \log^3 B)$.

In [9] Kaltoken has shown that the algorithm can actually be executed in $O(n^6 \log^2 B + n^5 \log^3 B)$ with a slighty weaker notion of reduced basis (Siegel's $\sqrt{2}$-reduction [42]): with condition (2) replaced by

$$2|b_i^*|^2 \geq |b_{i-1}^*|^2 \quad for \quad 1 < i \leq n. \tag{5}$$

This is not a loss of generality, indeed, both (2) and (5) lead to the following lemma,

Lemma 1 ([18]) *If L is a lattice with reduced basis $b_1, b_2, \ldots, b_n$, then for every nonzero $x \in L$,*

$$2^{\frac{n-1}{2}} |x| \geq |b_1|.$$

Schönage [34] then successfully attempted to change one basic choice of L^3 , let us first recall that a L^3 execution produces a somewhat bubble-sort sequence of the two following transformations,

Swap. Choose i, $1 < i \leq n$, such that (2) is not satisfied and interchange b_{i-1} and b_i.

Translation. Choose i and j, $1 \leq j < i \leq n$, such that (1) is not satisfied and translate b_i respectively to b_j: if r is the integer nearest to μ_{ij} replace b_i by $b_i - rb_j$.

It is well known [19] that independently of the order in which swaps and translations are applied and independently of the related choices of i and j, this lead after a finite number of steps to a reduced basis. In [18] always the smallest i is chosen for the swaps. Schönage has shown that a different strategy may lead to better results. Applying block reduction i.e. with i varying in small intervals for the swaps, he satisfies the following semi-reduction condition:

$$2^{n+i-j}|b_i^*|^2 \geq |b_j^*|^2 \quad for \quad 1 \leq j < i \leq n, \tag{6}$$

using at most $O(n^5 \log^3 B)$ bit operations.

A different approach has been investigated by Schnorr [31, 33]. In order to avoid exact computations on big integers, he developed a self-correcting algorithm which simulates the L^3 reduction through high precision floating-point arithmetic. His algorithm uses at most $O(n^5 \log B)$ arithmetic operations on $O(n + \log B)$ bit integers. Since this improvement is entirely different than Schönage's one, the two can certainly be combined to give a reduction algorithm using at most $O(n^{5\ 5} \log B + n^{3.5} \log^3 B)$ bit operations.

In a more experimental point of view, working on the factorization of polynomials, Abbott [1] has reported improved reductions. Beyond a tentative with floating-point arithmetic and one for a generalization of Lehmer's idea for integer gcd [13], he has given a method based on complete reductions of pairs of vectors and on postponed updating of the data. A series of experiments shows that this last method leads to low running times.

Parallel point of view. All the previous studies have been done in a sequential framework. Furthermore, none of them seems to be helpful to derive the parallel complexity of basis reduction, or even to design an efficient parallel algorithm. Anyway, to our knowledge, except von zur Gathen's classification [43], no work has been done before on the reduction from a parallel point of view. The computational costs of the sequential methods presented above, all include the factor $\mathcal{S} = O(n^2 \log_t B)$ which is an upper bound on the number of swaps that are necessary to reduce a basis [18]. Our empirical study of the L^3 complexity will show that whereas this bound is too pessimistic for subset sum problems, lattices for which the bound is experimentally reached can be constructed. A main problem from a parallel point of view will thus be to reduce this factor. We will also discuss the influence of the value of t in (2).

However, to decide whether the problem of computing a reduced basis, called *SHORT VECTORS* in [43], have a fast parallel algorithm is still an open question [43]. We refer to Cook [3] for the definitions of the complexity classes and of the $\mathcal{NC}$-reductions. Lattice basis reduction and gcd computations are strongly related, *GCD OF TWO INTEGERS* has been shown to be $\mathcal{NC}^1$-reducible to *SHORT VECTORS in dimension two* by von zur Gathen. But to decide whether the former problem is in $\mathcal{NC}$ is

also an open question [3], even if a sublinear algorithm has been given in [12]. Another related problem is *HERMITE FORM OVER Z*. As previously, *GCD OF TWO INTEGERS* is easily shown to be $\mathcal{NC}^1$-reducible to *HERMITE FORM OVER Z in dimension two* [10]. And in addition, the Hermite normal form viewed as an application of the basis reduction in [35] shows that *SHORT VECTORS* is a more difficult problem. We may finally remark that size-reducing a basis (to satisfy the condition (1)) is merely an Hermite normal form computation.

We are now ready to introduce our parallel approach to basis reduction. Evidently, a naive parallel implementation can be based on the parallelization of each swap of pair of vectors. This idea allows to use $O(n)$ processors, but let unchanged the value $O(n^2 \log_t B)$ for $\mathcal{S}$. Our main result is a parallel algorithm based on an heuristic which consists in swapping up to $O(n)$ pairs simultaneously. We show that this algorithm efficiently uses up to $O(n^2)$ processors. While the previous papers on reduction was using the same strategy to satisfy the condition (1) of weak reducedness, under the contraints of parallelism we use a different one. Recall that, as noticed by Lovász [20], this condition of weak reducedness is only partially used in order to establish the lemma 1: only $|\mu_{i+1,i}| \leq \frac{1}{2}$ is needed. The full strentgh of weak reducedness principally restrains the growth of the numbers on which operations are performed during the reduction.

2 L^3 complexity study

Instead of considering the total execution times of the reductions, we focus on the number of swaps which are performed during those reductions. This will lead to better conclusions concerning the L^3 complexity. For most of the lattices which are used by the applications, the theoretical upper bound $\mathcal{S} = O(n^2 \log_t B)$ on the number of swaps using the L^3 strategy is too pessimistic.

n	20	30	40	46
	193, 391	289, 834	386, 1288	444, 1523
n	50	56	60	66
	482, 1768	540, 2102	579, 2220	636, 2495

Table 1: $\log_{2/\sqrt{3}} B$ and *number of swaps* for subset sum n-lattices.

A first example is given by the basis associated to the fifth polynomial factorized by Lenstra [16]. For $n = 9$ and $\log_{2/\sqrt{3}} B \approx 1000$, 309 swaps are sufficient to reduce the basis. In the same way, we have run systematic tests for random instances of the subset sum problem [15, 24] (with randomly distributed solution of length $n/2$, and a density $d = 1/2$). The results are gathered in table 1. It can be verified that for the chosen subset sum lattices, with $\log B = O(n)$, we have $\mathcal{S} = O(n)$ instead of $O(n^3)$. Nevertheless, we can construct lattices which experimentally behave like the worst case of L^3 .

n	4	6	8	10
	96, 25	163, 109	193, 291	241, 678
n	12	14	18	22
	290, 1347	352, 2518	434, 6471	533, 13280

Table 2: $\log_{2/\sqrt{3}} B$ and *number of swaps* for dense n-lattices ($\kappa = 8$).

Let $F_{i,j}, 1 \leq i,j \leq n, i \neq j$ be the $n \times n$ matrix which transform any matrix by left multiplying its rows i and j, in this order, by the 2×2 matrix $F = \begin{bmatrix} 2 & 1 \\ 1 & 0 \end{bmatrix}$. Matrices $F_{i,j}$ are used to derive the worst case of the reduction in dimension 2 [41]. They may be combined in many ways to give lattices of small determinant which rows are long vectors, say *dense lattices*. A simple way to proceed is as follow. We define the matrices $W_{n,\kappa}$:

$$W_{n,\kappa} = \prod_{j\,odd \geq 1} \prod_{i\,odd \geq 1} F^{\kappa}_{i,i+j}, \tag{7}$$

where we substract n to $i+j$ if it is necessary to fit the dimension of the matrices. From the eigenvalues $1+\sqrt{2}$ and $1-\sqrt{2}$ of F, it can be derived that:

$$\| W_{n,\kappa} \|_\infty = O((1+\sqrt{2})^{n\kappa}). \tag{8}$$

By reducing the lattices given by the matrices $W_{n,8}$ we have obtained the results of table 2. Let us first notice that $\mathcal{S} = O(n^3)$. Furthermore, $\mathcal{S}$ tends to behave like $\frac{n^2}{2} \log_{1+\sqrt{2}} B$.

n	14	20	26	30
ratio	1.53	1.88	2.69	2.73
n	36	40	50	60
ratio	3.2	3.59	4.15	4.5

Table 3: *number of swaps* for $t = 1$ over *number of swaps* for $t = 3/4$ for subset sum n-lattices.

n	4	6	8	10
ratio	1.08	1.07	1.20	1.28
n	12	14	16	18
ratio	1.49	1.64	1.81	1.92

Table 4: *number of swaps* for $t = 1$ over *number of swaps* for $t = 3/4$ for dense n-lattices ($\kappa = 8$).

However, to determine exactly the worst case of L^3 is still a major problem, possibly related to the one of proving that L^3 still runs in polynomial time for $t = 1$. Lagarias and Odlyzko [15] had soon remarked that in practice a reduction takes about three times for $t = 1$

as long as for $2/\sqrt{3}$. To finish this empirical study of L^3 complexity, we precisely compare in table 3 and in table 4, the number of swaps in the two cases: $2/\sqrt{3}$ and $t = 1$. The major remark we can do is that for two definitely different types of lattices, the ratios of the number of swaps for $t = 1$ over the number of swaps for $2/\sqrt{3}$ increase like $O(n)$ with similar slopes.

3 L^3 -like reductions

For a better understanding, this section is dedicated to a presentation of the sequential version of the method we will use for our second parallel algorithm. The order in which the swaps are applied in the *algorithm 1* would be too restricting in a parallel framework. The algorithm may be easily modified in order to allow any strategy: this lead to *algorithm 2* below which performs say an *any swap reduction.*

Algorithm 2 : any swap reduction

Input data : $b_1, b_2, \ldots, b_n$ a basis
Compute the $b_1^*, b_2^*, \ldots, b_n^*$ and the μ_{ij}
for i = 2 **to** n
  for j = i-1 **to** 1
    if $|\mu_{ij}| > \frac{1}{2}$ **then** $b_i := b_i - \lceil \mu_{ij} \rfloor b_j$
while the basis is not reduced
  (F) find i **s.t.** $|b_i^* + \mu_{i,i-1} b_{i-1}^*|^2 t^2 < |b_{i-1}^*|^2$
  swap b_i and b_{i-1}
  for j = i-1 **to** 1
    if $|\mu_{ij}| > \frac{1}{2}$ **then** $b_i := b_i - \lceil \mu_{ij} \rfloor b_j$

Output data : $b_1, b_2, \ldots, b_n$ a t-reduced basis.

We show in the following that a complexity study similar to the one used for the L^3 reduction can be applied for the *any swap reduction.*

Remark (Coefficient growth). Let us already notice that during the translations, the integers will have length as large as $O(n^2 \log B)$ (versus $O(n \log B)$ for the L^3 reduction). This could appear to be a major drawback. Contrary to that, on the one hand, this seems to be necessary in order to decrease the parallel complexity (see section 4.3). On the other hand, empirical results show that coefficients are not so big, the corollary 2 below on *particular lattices* will prove this fact.

Our arguments are those laid out in [18] with one modification given by the the following lemma.

Lemma 2 *If $1/\beta$, $\beta \in \boldsymbol{Z}$ is a lower bound for the initial values of the $|b_i^*|^2$, $1/\beta$ remains a lower bound throughout the algorithm.*

Proof. Let us first notice that each weak reduction does not modify the value of the $|b_i^*|^2$. As for the new values of the norms after a swap $(i, i-1)$, say b'_{i-1} and b'_i, we have:

$$\begin{cases} b'^2_{i-1} = |b_i^*|^2 + \mu_{i,i-1}^2 |b_{i-1}^*|^2 > |b_i^*|^2, \\ b'^2_i > t^2 |b_i^*|^2 > |b_i^*|^2. \end{cases} \quad (9)$$

The assertion of the lemma follows directly. □

Proposition 1 (Complexity study) *The number of arithmetic operations needed for the any swap reduction is $O(n^4 \log B)$. If $1/\beta$ is a lower bound for the initial values of the $|b_i^*|^2$, the integers on which these operations are performed are of binary length $O(n \log n\beta B)$.*

Proof. We know that throughout the algorithm, $|b_i^*|^2 \leq B^2$ for $1 \leq i \leq n$. It remains to estimate $|b_i|^2$ and μ_{ij}^2. At label (F), $|b_i|^2 \leq nB^2$: it is true at the beginning of the algorithm, it trivially remains true after a swap and it can be established [18] after a weak reduction with $|\mu_{ij}| \leq \frac{1}{2}$. For the Gram-Schmidt coefficients, we have initially, $|b_i^*|^2 \geq 1/\beta$, therefore, applying the lemma, we know that this remains true during the algorithm. We now deduce that at label (F) and during a swap, $\mu_{ij}^2 < n\beta B^2$:

$$\begin{aligned} nB^2 \geq |b_i|^2 = |b_i^*|^2 + \textstyle\sum_{l=1}^{i-1} \mu_{il}^2 |b_l^*|^2 \\ > \mu_{ij}^2 |b_j^*|^2 \geq 1/\beta. \end{aligned} \quad (10)$$

Following the idea of Kaltofen [9] we know that during a weak reduction the b_i can be computed modulo $\sqrt{n}B$, and it can be shown by induction that the $|\mu_{ij}|$ stay of binary length $O(n \log n\beta B)$. This last bound instead of $O(n + \log n\beta B)$ for the L^3 reduction: coefficients $|\mu_{kj}|$, $1 \leq k < i$ which are needed to reduce $|\mu_{ij}|$ are not necessary lower than 1/2 but only bounded as $O(n\beta B^2)$.

The conclusion then comes immediately: at each of the $O(n^2 \log_t B)$ steps, the algorithm performs at most $O(n^2)$ arithmetic operations on integers of binary length $O(n \log n\beta B)$. □

Corollary 1 (General case) *The integers on which the operations are performed during the any swap reduction are of binary length $O(n^2 \log B)$.*

Proof. Let d_i, $1 \leq i \leq n$, denote the Gram's determinant of the lattice generated by $b_1, b_2, \ldots, b_i$. We have initially,

$$|b_i^*|^2 = \frac{d_i}{d_{i-1}} \geq \frac{1}{B^{2i}} \geq \frac{1}{B^{2n}}, 2 \leq i \leq n. \quad (11)$$

It suffices to apply the proposition with $\beta = B^{2n}$. □

For most of the lattices used by the applications (for instance those cited in the introduction) the proposition

can be applied with much better values of β. Especially, for the subset sum lattices we have the following corollary.

Corollary 2 (Particular lattices) *The integers on which the operations are performed during the any swap reduction of a subset sum lattice are of binary length* $O(n \log nB)$.

Proof. Such lattices are given by a matrix of the form:

$$L = \begin{bmatrix} 1 & 0 & \dots & 0 & -a_1 \\ 0 & 1 & \dots & 0 & -a_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & -a_n \\ 0 & 0 & \dots & 0 & M \end{bmatrix}. \quad (12)$$

The Gram-Schmidt orthogonalization of L leads to the relations:

$$\begin{cases} B^2 \geq |b_i^*|^2 \geq 1, \ 1 \leq i \leq n-1, \\ B^2 \geq |b_n^*|^2 \geq 1/nB^2. \end{cases} \quad (13)$$

Looking at the $\mu_{i,j}^2$, we may derive that,

$$nB^2 > \mu_{ij}^2 |b_j^*|^2 \geq \frac{\mu_{ij}^2}{nB^2}. \quad (14)$$

We can take $\beta = n^2 B^4$ in the proposition. □

This also proves that throughout the algorithm but during the weak reductions, the bit length of the numerators and denominators of the rationals are bounded as $O(\log n + \log B)$. This remark could be useful for methods based on floating-point arithmetic.

4 Parallel basis reduction

We now present two parallel algorithms for basis reduction and study their complexities. The first one is a direct parallelization of the sequential L^3 algorithm: we will point out the limits of such an approach. The second one is based on an original idea of simultaneous swaps of the vectors. We will not discuss the parallelization of the Gram-Schmidt process which is of lower cost.

Our abstract-machine model consists in P processors. The network topology is a ring or a 2-dimensional torus. Each processor has a private memory an can communicate by a message passing protocol with its neighbours. We assume that the binary cost for communicating an integer of length $O(l)$ is also bounded as $O(l)$. The machine operates in an asynchronous MIMD mode and the communications are by rendez-vous.

4.1 L^3 parallel reduction

The first parallel algorithm is the parallel version of the original L^3 reduction (algorithm 1). It is simply based on the parallelization of each swap of pair of vectors and of each translation, using a ring topology. The number of processors is given by $P = \gamma n$, $0 < \gamma \leq n$. The processors are numbered from 0 to $P-1$ (we operate modulo P on those numbers). The $\mathcal{S}$ steps of the sequential reductions are preserved.

Data repartition. The three matrices which give the vectors b_i and b_i^*, and the coefficients μ_{ij} have to be distributed among the processors. Simply by noticing that each swap (b_{i-1}, b_i) involves $O(n)$ coefficients whilst each translation involves $O(ni)$ coefficients, we decide to minimize the communication cost of the translations. Precisely, a distribution of the rows of the matrices (thus of the vectors of the basis) would induce O(ni) data transfer for each of these translations. Our solution consists in a column distribution of the matrices: it limits the number of data transfer to $O(i)$ for each translation. The column j of the matrices (thus the coordinate j of the vectors) is allocated to the processor $alloc(j)$. Furthermore, we assume that if $j = j' \pm 1$ then $alloc(j) = alloc(j') \pm 1$.

Communications. At each step i, the processors $alloc(i-1)$ and $alloc(i)$ take the decision to swap or to translate and inform the other processors. After that, a swap will have a communication cost bounded as $O(n^2 \log B)$: only two neighbour processors need to communicate. They exchange at most $O(n)$ coefficients of bit length $O(n \log B)$. Concerning the translations, let us detail a weak reduction of the vector b_i. For $j < i$, the processor $alloc(j)$ has to send r_j (the integer nearest to μ_{ij}) to every processors. This can be achieved by using either a broadcast or a pipeline strategy. Many parallel algorithms use these techniques, especially Gaussian elimination. The reader will refer to [4, 25, 30] for detailed explanations. The cost is also bounded by the cost of n transfer of a coefficient of length $O(n \log B)$.

Arithmetic. Following the previous authors concerning the broadcast or pipeline techniques, we equidistribute the calculus by chosing a wrap repartition i.e. $alloc(j) \equiv j \ mod \ P$. Consequently, for each swap each processor performs at most $O(n)$ arithmetic operations on coefficients of length $O(n \log B)$, and at most $O(n^2/P)$ for each translation.

Proposition 2 *Using $O(n)$ processors, a basis may be reduced in $O(n^5 \log^3 B)$ binary arithmetic steps and $O(n^4 \log^2 B)$ binary communication steps.*

This assertion directly comes from the communication and arithmetic costs study (using classical arithmetic

procedures), and from $\mathcal{S} = O(n^2 \log_t B)$. □

Both this algorithm and the next one have been implemented under the *PAC project* [38, 26]. The performance results will be discussed in section 5.

Remarks. Each arithmetic operation could also be executed on many processors. It is shown in [29] that the algorithm would run in $O(n^3 \log^2(nB))$ binary arithmetic steps on $O(n^2 \log^2(nB))$ processors. But because of the communication cost, such an approach is not thinkable for a practical algorithm.

To use either the original L^3 reduction or one of the improved algorithms presented in the introduction will always preserve the factor $\mathcal{S}$ in the complexity. The main problem in a parallel point of view is thus to bypass this difficulty.

4.2 Heuristic for parallel reduction

The basic idea we use for the second algorithm is that probably several swaps may be done simultaneously at each step of the reduction. We can formulate the following heuristic for a parallel basis reduction,

Heuristic 1 (All swaps reduction) *We use P processors, $1 \leq P \leq n$. A basis can be reduced in $\mathcal{T}$ phases, each consisting in at most $P/2$ simultaneous swaps among all the possible swaps (b_i, b_{i+1}) for i odd, next $P/2$ simultaneous swaps among all the possible other swaps (b_{i-1}, b_i). Each phase clearly consists in at most $(n-1)$ simultaneous swaps thus $\mathcal{T}$ satisfies in the worst case:*

$$\mathcal{S}_{min} = (n/2) \log_t B \leq \mathcal{T} \leq \mathcal{S}. \qquad (15)$$

Let $\sigma(k)$, $1 \leq k \leq \mathcal{T}$, be the number of swaps performed at the phase k. We do not yet know how to show, in particular situations, that $\sigma(k)$ is strictly greater than 1 and/or near to P. This would justify our heuristic and above all, give the complexity of our parallel algorithm.

However, we can give some basic empirical observations on $\sigma(k)$. Using the test lattices of the section 2 we have obtained the results presented in the tables 5,6,7. We first compare the empirical average number of swaps per phase (say *actual*) to P which is the theoretical largest possible number of swaps per phase (say *max*). We have thus measured the ratio

$$actual/max \; = \; [\sum_{k=1}^{\mathcal{T}} \sigma(k)] \; / \; \mathcal{T}P. \qquad (16)$$

Always many swaps can be done in parallel.

max	4	6	15	20	30
actual/max	0.98	0.97	0.85	0.87	0.77

Table 5: *number of swaps* per phase for subset sum 60-lattices (section 2).

max	4	6	8	12
actual/max	0.94	0.88	0.87	0.74

Table 6: *number of swaps* per phase for dense 12-lattices (section 2).

Our main observation comes from table 7. For our test lattices, the average value of $\sigma(k)$ using $n/2$ processors is nearly equal to $2n/3$. That is to say, the heuristic leads to the empirical value $\mathcal{T} = O(n \log B)$ in the worst case and to an empirical efficiency of 2/3.

n	12	20	28	36	40
$E(\sigma(k))$	8.4	13	16	23.4	26

Table 7: the *average number of swaps per phase* as a function of n for subset sum n-lattices (section 2).

It is finally important to remark that using a *all swaps reduction* we generally perform less swaps than using the L^3 reduction. For instance, a test with a subset sum 60-lattice have asked for 1347 swaps using the L^3 strategy while only about 1100 swaps was needed using a *all swaps strategy*. In the same way for a dense 12-lattice we have obtained 2220 swaps using L^3 versus about 1500.

4.3 All swaps parallel reduction

The *all swaps parallel reduction* algorithm is based on the previous heuristic. We will now use a 2-dimensional torus in order to benefit both of simultaneous swaps and of the parallelization of each vector operations (swaps and translations). The number of processors is given by $P = \gamma n \times \delta n$, $0 < \gamma, \delta \leq n$ and $\gamma \geq \delta$. The processors are labelled (p_h, p_v) with p_h from 0 to $P_h = \gamma n - 1$ in the first dimension, say *horizontal*, and p_v from 0 to $P_v = \delta n - 1$ in the second dimension, say *vertical*.

Data repartition. We proceed as in section 4.1 by first distributing the columns of the matrices according to the horizontal dimension of the torus. Each of the p_v horizontal rings will thus manage n/p_v rows (vectors of the basis) and perform the associated operations. In particular, p_v translations will be feasible simultaneously. In the same way, we distribute the rows of the matrices according to the vertical dimension. Consequently, each of the p_h vertical rings will manage n/p_h column operations during the swaps. Like above, $p_h/2$ swaps will be done in parallel. Using the *wrap repartition* of section 4.1 in one dimension, we have in two dimensions for a coefficient (i,j),

$alloc((i,j)) = (alloc(i), alloc(j))$. Furthermore, the sub-diagonal elements $\mu_{i-1,i}$ are duplicated: they are also stored and updated by the processors $(p_h, 0)$ of the first horizontal ring in order to take the swap decisions.

The parallel reduction. The heuristic directly leads to the parallel algorithm which consists in swapping as much as possible in parallel as long as the basis is not reduced. Let us give an overall description of each of the $\mathcal{T}$ phases of this iterative process. A phase implements P simultaneous *any swap reductions* of section 3. Six main successive steps may be distinguished:

1. The processors of the first horizontal ring (labelled $(p_h, 0)$) pairwise exchange the data necessary to the searches and to the swap decisions: processor $(p_h, 0)$ with processor $(p_{h+1}, 0)$ then with processor $(p_{h-1}, 0)$.
2. A total exchange communication allows every processor of the torus to know all the swaps of the phase.
3. The swaps are performed. Communications on the horizontal rings allow the column operations, afterwards, communications on the vertical rings allow the row operations. From the data allocation function, those communications simply consists in swaps between neighbouring processors.
4. The columns of the matrices are vertically duplicated. At the end of this step, every processor (p_h, p_v) of a given vertical ring stores the complete columns j with $alloc(j) = p_h$.
5. The previous duplication allow to perform the translations in parallel using a *broadcast* or a *pipeline strategy* on every horizontal ring.
6. End detection: after a total exchange communication, every processor knows if it still remains any swap to perform.

Before the complexity study, let us give a simplified version of the algorithm.

Algorithm 3 : all swaps reduction
For the processor (p_h, p_v)

`Input data` : $b_1, \ldots, b_n$ a basis, the $b_1^*, \ldots, b_n^*$ and the μ_{ij}
`while` the basis is not reduced
- `if` $(p_v = 0)$
  `then` pairwise searches and swap decisions involving vectors b_i with $alloc(i) = p_h$.
  `else` wait for the swap decisions
- swap
- duplicate the columns j with $alloc(j) = p_h$ communicating with processors $(p_h, .)$
- weak reduce the swapped rows i with $alloc(i) = p_v$ communicating with processors $(., p_v)$
- `if` $(p_v = 0)$
  `then` take and send a local end decision
  `else` wait for the end decision

`Output data` : $b_1, b_2, \ldots, b_n$ a t-reduced basis.

Communications. The algorithm widely uses the *total exchange* (also called *all to all*) communication pattern. Many authors have worked on the subject in mesh topologies, the reader may refer to some recent papers [5, 36, 22] to get a good insight. For our asymptotic analysis (communications are of a lower cost compared with arithmetic) we just need an easy result: p_v processors need $O(p_v nl)$ binary communication steps in order to exchange n integers of binary length l. The two major communication costs come from step 4 and step 5 of each phase. At step 4 every processors stores $n^2/(p_h p_v)$ cœfficients of length $O(n \log B)$ that are to be exchanged. Since the vertical rings are of length p_v, $O(n^2/p_h)$ steps or $O(n^3 \log B/p_h)$ binary steps are necessary. Concerning the step 5, at most $p_h/2$ swaps has been done, equivalently at most $p_h/2$ weak reductions has to be performed. Since p_v such reductions can be done simultaneously using a *pipeline strategy* (see section 4.1) on each horizontal ring, it can be done in $O(np_h l/p_v)$ binary steps. The length l will be either $O(n^2 \log B)$ or $O(n \log nB)$ (see the previous section and the proposition below).

Arithmetic. From the choice of the allocation fuction of the matrices, the computations are equidistributed. The major cost still comes from the weak reductions (step 5). We have seen that each phase induced at most $p_h/2$ such reductions. Since they are distributed on the horizontal rings, p_v reductions can be performed in parallel. Each reduction ($O(n^2)$ operations on integers) being itself distributed on p_h processors, we get $O((p_h/p_v) \times (n^2/p_h))$, that is $O(n^2/p_v)$ arithmetic operations on integer per phase. For integers of length l we finally have $O(n^2 l^2/p_v)$ for the binary arithmetic complexity (using classical arithmetic procedures).

Proposition 3 *Using $O(n^2)$ processors and assuming that the integers involved in the computations are at most of length $O(l)$, each phase of the all swaps parallel reduction needs at most $O(nl^2)$ binary arithmetic steps and $O(nl)$ binary communication steps. In particular, for the subset sum lattices or those used by the applications, the arithmetic binary complexity is $O(n^3 \log^2 nB)$. This lead to $O(n^4 \log^3 nB)$ for the whole reduction if $\mathcal{T} = O(n \log B)$.*

The assertion can be directly deduced from the previous study with $p_h = p_v = n$, and from the corollary 2 concerning the particular lattices. □

Remark. If $O(n^4 \log^2(nB))$ processors are available, it is shown in [29] that each phase of the all swaps reduction runs in $O(n \log^2(n \log B))$.

5 Performance results

We have implemented the two algorithms L^3 *parallel reduction*, say L^3-PR, and *all swaps parallel reduction*, say AS-PR. This work has been done in the frame of the *PAC project* [27, 26, 38]. Let us briefly recall that the target machine is a *Méganode* [38] based on 128 Inmos transputers T800. Each processor has a 1Mb private memory and communicates by rendez-vous with at most four neighbours. This makes the *Méganode* exactly correspond to our abstract model.

As noticed by several authors (see [1] for instance) a closer look at the algorithms shows that we only need integer arithmetic. This has been applied for our implementations. As usual let us now discuss the measured speed-ups of the algorithms: the ratios of the sequential execution time over the execution times on P processors. Even in an empirical point of view, the cases $P \ll n$ are definitely not interesting: the speed-ups are nearly to P for both algorithms and no conclusion can be made concerning the complexity. In the case $P \approx n$ for a subset sum 36-lattice, the speed-ups of L^3-PR stay lower than 6. Those bad performances are essentially due the communication cost which tends to predomine as P increases. Consequently, although its implementation is easy, the algorithm L^3-PR presents no interest but for a small number of processors.

p_h	1	6	20	30
$p_v = 1$	1	5.3	18.4	23.9
$p_v = 2$	1.4	9.2	32.5	52
$p_v = 3$	1.6	11.3	43.7	59.6

Figure 8 : AS-PR, *speed-ups* for subset sum 60-lattices.

Contrary to that, AS-PR on a similar subset sum 36-lattice leads to a speed-up of 17.4 on a ring of 18 processors (case $P \approx n$). And the table 8 summarizes the good results obtained with AS-PR in the case $P > n$. It can be seen that on a 90 ($p_h = 30$, $p_v = 3$) processors torus an efficiency of 65% is obtained for the dimension 60. To motivate the conclusion and our further studies, we may finally precise that the limits of AS-PR are reached for large values of p_v. This is mainly due to a bad distribution of the arithmetic computations: the operations on infinite precision integers are not appropriate to a static mapping.

6 Conclusion

The *all swaps parallel reduction* that we propose leads to the time bound $O(n^2 \log B \log^2(n \log B))$ for the reduction of "good" lattices of dimension n. Furthermore, the basic idea we have used in order to achieve this bound, can be easily combined to the improved sequential methods cited in the introduction and give better results. The implementations are still under test and development. Studies on *dynamic load-balancing* [28] should permit to go beyond the current limits imposed by the static mapping of the tasks on the processors.

Note added on April 27: We have obtained the time bound $O(n \log B \log^2(n \log B))$.

References

[1] J.A. Abbott. *On the factorization of polynomials over algebraic fields.* PhD thesis, University of Bath, 1988.

[2] L. Adleman. On breaking the iterated Merkle-Hellman public key cryptosystem. In *15th ACM Symposium on Theory of Computing, pp 402-412*, 1983.

[3] S.A. Cook. The classification of problems which have fast parallel algorithms. In *Int. Conf. Foundations of Computation Theory, Borgholmm 1983, Lect. N. Comp. Sc. 158 pp 78-93*, 1983.

[4] M. Cosnard, , B. Tourancheau, and G. Villard. Gaussian elimination on message passing architectures. In *Supercomputing, E.N. Houtis et al. eds, LNCS 297*, 1988.

[5] P. Fraigniaud. *Communications intensives sur architectures à mémoire distribueé.* PhD thesis, ENS-Lyon, 1990.

[6] A.M. Frieze, R. Kannan, and J.C. Lagarias. Linear congruencial generators do not produce random sequences. In *25th IEEE Symposium on Theory of Computing, pp 480-484*, 1984.

[7] J. Hastad, B. Helfrich, J.C. Lagarias, and C.P. Schnorr. Polynomial time algorithms for finding integer relations among real numbers. In *3rd Symposium on Theoretical Aspects of Computer Science, Lect. N. Comp. Sc.*, volume 210, pages 105–118, 1986.

[8] J. Hastad, B. Just, J.C. Lagarias, and C.P. Schnorr. Polynomial time algorithms for finding integer relations among real numbers. *SIAM J. Comput.*, 18(5):859–881, 1989.

[9] E. Kaltofen. On the complexity of finding short vectors in integer lattices. In *Proc EUROCAL 83 (London) Springer LNCS 162 pp 236-244*, 1983.

[10] E. Kaltofen, M.S. Krishnamoorthy, and B.D. Saunders. Fast parallel computation of Hermite and Smith forms of polynomials matrices. *SIAM J. Alg. Disc. Meth.*, 8 4, pp 683-690, 1987.

[11] R. Kannan, H.W. Lenstra, and L. Lovász. Polynomial factorization and nonrandomness of bits of algebraic and some transcendental numbers, 1984. Carnegie-Mellon University Cs-84.

[12] R. Kannan, G. Miller, and L. Rudolph. Sublinear parallel algorithm for computing the greatest common divisor of two integers. *SIAM J. Comput.*, 16 1, 1987.

[13] D.E. Knuth. *Seminumerical algorithms.* Addison-Wesley, 1981.

[14] J.C. Lagarias. The computational complexity of simultaneously diophantine approximation problems. In *23rd IEEE symposium FOCS, pp 32-39*, 1983.

[15] J.C. Lagarias and A. Odlyzko. Solving low-density subset sum problems. *Journal of ACM*, 32 1 pp 229-246, 1985.

[16] A.K. Lenstra. Lattices and factorization of polynomials over algebraic number fields. In *Proc EUROCAM 82 (Marseille) Springer LNCS 144 pp 32-39*, 1982.

[17] A.K. Lenstra. Factoring polynomials over algebraic number fields. In *Proc EUROCAL 83 (London) Springer LNCS 162 pp 245-254*, 1983.

[18] A.K. Lenstra, H.W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Math. Annalen*, 261 pp 513-534, 1982.

[19] H.W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8 4, pp 538-548, 1983.

[20] L. Lovász. *An algorithmic theory of numbers, graphs and convexity.* CBMS-NSF Regional Conferences Series in Applied Mathemetics, SIAM, 1986.

[21] V. Niemi. A new trap-door in knapsacks. In *Advances in Cryptology, EUROCRYPT '90, LNCS 473 pp 405-411, Springer-Verlag*, 1991.

[22] B. Plateau and A. Touzène. Optimal multinode broadcast on a mesh connected graph with reduced bufferization. In *Second European Conference on Distributed Memory Computing. Munich, FRG*, 1991.

[23] M. Pohst. A modification of the LLL reduction algorithm. *Journal of Symbolic Computation*, 4, pp 123-127, 1987.

[24] S.P. Radziszowski and D.L. Kreher. Solving subset sum problems with the LLL algorithm. In *Third SIAM Conference on Discrete Mathematics, Clemson, SC.*, 1986.

[25] Y. Robert, B. Tourancheau, and G. Villard. Data allocation Strategies for the Gauss and Jordan Algorithms on a Ring of Processors. *Information Processing Letters*, 31, 1989.

[26] J.L. Roch. The PAC System and its Implementation on Distributed Architectures. In *Computer with Parallel Architectures : T. Node, ed. D. Gassilloud, J.C. Grossetie, Kluwer Ac. Pub.*, 1991.

[27] J.L. Roch, F. Siebert, P. Sénéchaud, and G. Villard. Computer Algebra on a MIMD machine. *ISSAC'88 and in SIGSAM Bulletin, ACM*, 23/11, p.16-32, 1989.

[28] J.L. Roch, A. Vermeerbergen, and G. Villard. Load-balancing for algebraic computations, 1992. Submitted to CONPAR92.

[29] J.L. Roch and G. Villard. Parallel gcd and lattice basis reduction, 1992. Submitted to CONPAR 92.

[30] Y. Saad. Gaussian elimination on hypercubes. In *Parallel Algorithms and Architectures, Eds M. Cosnard et al., North-Holland*, 1986.

[31] C.P. Schnorr. A more efficient algorithm for lattice basis reduction. *Journal of Algorithms*, 9, pp 47-62, 1988.

[32] C.P. Schnorr. Factoring integers and computing discrete logarithms via diophantine approximation. In *EUROCRYPT'91, Brighton-U.K., LNCS 547*, pages 281–293, 1991.

[33] C.P. Schnorr and M. Euchner. Lattice basis reduction: improved practical algorithms and solving subset sum problems. In *FCT91, Lect. Nt. Comp. Sc.*, 1991.

[34] A. Schönhage. Factorization of univariate integer polynomials by diophantine approximation and an improved basis reduction algorithm. In *11th ICALP, Lect.Nt.Comp.Sc. 172, Springer*, 1984.

[35] A. Schrijver. *Theory of linear and integer programming.* Wiley-Interscience series in Discrete Mathematics, 1986.

[36] D.S. Scott. Efficient all to all communication patterns in hypercube and mesh topologies. In *The 6th Distributed Memory Computers Conference. Portland, Oregon USA*, 1991.

[37] A. Shamir. A polynomial time algorithm for breaking the Merkle-Hellman cryptosystem. In *23rd IEEE Symposium FOCS*, 1982.

[38] F. Siebert and G. Villard. PAC : First experiments on a 128 transputers Meganode. In *International Symposium on Symbolic and Algebraic Computation, Bonn Germany*, 1991.

[39] J. Stern. Secret linear congruencial generators are not cryptographically secure. In *28th IEEE Symposium FOCS*, 1987.

[40] J. Stern and P. Toffin. Cryptanalysis of a public-key cryptosystem based on approximations by rational numbers. In *Advances in Cryptology, EUROCRYPT '90, LNCS 473 pp313-317, Springer-Verlag*, 1991.

[41] B. Vallée. *Une approche géométrique de la réduction des réseaux en petite dimension.* PhD thesis, Université de Caen, 1986.

[42] B. Vallée. La réduction des réseaux. Autour de l'algorithme de Lenstra, Lenstra, Lovász. *Theoretical Informatics and Applications*, 23 3, pp 345-376, 1989.

[43] J. von zur Gathen. Parallel algorithms for algebraic problems. *SIAM J. Comput.*, 13 4, pp 802-824, 1984.