Research on the Parallelization of LLL Algorithm

Liu Xianghui[1, 2], Wang Zheng[1, 2], Quan Jianxiao[2]
[1]Department of Applied Mathematics, Zhengzhou Information Science and Technology Institute, Zhengzhou 450002, China
[2]State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450002, China
e-mail: lxhkz2002@163.com

***Abstract*—Lattice basis reduction algorithms are important tools in the area of cryptanalysis. LLL algorithm is one the most famous algorithms and the parallelization of LLL algorithm has been received increasing attentions. In this paper, the traditional block-based LLL algorithm is implemented in a parallel system and the efficiency is analyzed. With the analysis result, we indicate the bottleneck of the traditional block-based algorithm and give a new parallel implementation scheme of block-based LLL algorithm. The experimental results show that the new scheme behaves much better comparing to the old one.**

Keywords-lattice basis reduction; LLL algorithm; parallel implementation; block-based

I. INTRODUCTION

Lattice basis reduction, the problem of finding a basis of a lattice with "short" vectors, is an important problem in the geometry of numbers, which is applied widely used in the area of combinatorial optimization, computer algebra and algorithmic number theory. Also, lattice basis reduction algorithms were once of interest primarily to number theorists studying quadratic forms. However, since the 1980s when the LLL algorithm was invented [1], lattice basis reduction has been showing the tremendous power to cryptanalysis and becoming a new research hotspot in the field of cryptology. Firstly, the public key cryptosystem based on knapsack problem was cracked successfully by LLL algorithm in polynomial time [2]. Subsequently, LLL algorithm was widely used to attack RSA schemes under various additional assumptions, such as small exponent attack, partial key exposure attack and factoring the modulus when a certain portion of the bits of p are known in advance [3][4][5]. At the same time, LLL algorithm was also applied to factor large composite integers [6]. In a word, LLL algorithm has been playing an important role for the cryptanalysis.

LLL algorithm is first polynomial time algorithm for finding a basis of a lattice with "short" vector. Therefore, after its publication, LLL algorithm was immediately recognized as one of the most important lattice reduction algorithm and algorithmic achievements of the twentieth century because of its broad applicability and apparent simplicity. For a n-dimensional lattice, LLL algorithm can find a non-zero short vector in polynomial time and the length of the vector is at most $2^{(n-1)/2}$ times longer than the shortest vector. Nevertheless, the run time for sequential LLL algorithm for large dimension or with big entries is still quite high. For example, in order to factor integers that are 500 bits long the dimension of the lattice should be about 6300 and the entries contain integers that are 1500 bits long, which cannot be acceptable for LLL algorithm [7]. Thus, there is great interest in parallelizing LLL algorithm as to achieve an additional improvement in reducing the run times Backes introduced a new parallel variant of LLL algorithm suitable for multithread environment [8]. Experiments showed (compared to the non-parallel algorithm) a speedup factor of about 1.75 for the 2-thread and close to factor 3 for the 4-thread version. At the same time, Backes proposed some modified parallel versions on the subsections of LLL algorithm, such as GSO, size-reduction and so on. However, the algorithms of Backes were proposed in multithread environment requiring to sharing memory, which leads to the parallel expansibility being limited. In fact, as early as in 1998, Wetzel was used to propose the idea of block-based LLL algorithm [9], but he did not implement it on a parallel computer or for a distributed system.

In this paper, we study the parallelization scheme of block-based LLL algorithm and indicate the bottleneck of the traditional algorithm. Based on the analysis, we also propose a new parallel scheme for the block-based LLL algorithm. The paper is organized as follows. In section 2, we give a brief introduction to the theory of LLL algorithm. In section 3, we analyze the traditional algorithm and present a new parallel scheme. Section 4 we give the experimental results and section 5 is conclusions.

II. FUNDAMENTAL KNOWLEDGE OF LLL ALGORITHM

In this section, we give some fundamental knowledge of lattice basis reduction, LLL algorithm briefly. For further details we refer to [1][10].

Definition 1. Let $b_1, b_2, \cdots, b_m \in R^n$ be linearly independent vectors where R^n is n-dimensional vector space on real number field. Then $L(b_1, b_2, \cdots, b_m) = \{z = \sum_{i=1}^{m} \lambda_i b_i \mid \lambda_i \in Z\}$ is a lattice and $(b_1, b_2, \cdots, b_m)$ is a basis of the lattice. We call m the dimension of the lattice. If $m = n$, the lattice is full-rank. If all the vectors of the lattice are in the integral ring Z, namely $R = Z$, the lattice is called integral lattice.

For a *n*-dimensional vector $b=(b_1,b_2,\cdots,b_n)$, we denote $\| b \|=(b_1^2+b_2^2+\cdots+b_n^2)^{1/2}$ by its Euclidean norm.

Shortest vector problem (SVP) is to find a shortest nonzero vector given a lattice. It is has been shown that SVP is NP-hard under randomized reductions and there is no polynomial time algorithm for high dimensional lattices. Lattice basis reduction algorithms are the basic tools for solving SVP and LLL algorithm is the most important one of them. We describe the LLL algorithm as follows.

Definition 2. For a lattice L with a basis $b_1,b_2,\cdots,b_n \in R^n$, $b_1^*,b_2^*,\cdots,b_n^*$ is the corresponding Gram-Schmidt orthogonalization basis and μ_{ij} is the orthogonalization coefficients. We call the basis $b_1,b_2,\cdots,b_n$ a LLL reduced with parameter δ if the following conditions are satisfied:

(1) $|\mu_{ij}| \le \frac{1}{2}, 1 \le j < i \le n$;

(2) $\delta \| b_{k-1}^* \|^2 \le \| b_k^* \|^2 + \mu_{k,k-1}^2 \| b_{k-1}^* \|^2, k=2,\cdots n, \frac{1}{4} < \delta \le 1$.

Algorithm 1. LLL algorithm

Input: Lattice basis $b_1,b_2,\cdots,b_n \in Z^n$ and reduced parameter δ ($\frac{1}{4} < \delta \le 1$)

Output: a LLL reduced with parameter δ

(1) compute the Gram-Schmidt orthogonalization;
(2) $k=2$;
(3) while($k \le n$) do
(4) for($j=k-1; j \ge 1; j--$) do
(5) if($|\mu_{k,j}| > \frac{1}{2}$) then
(6) $b_k = b_k - \lceil \mu_{k,j} \rfloor b_j$, $\mu_{k,j} = \mu_{k,j} - \lceil \mu_{k,j} \rfloor$;
(7) fi
(8) od
(9) if($\delta \| b_{k-1}^* \|^2 > \| b_k^* \|^2 + \mu_{k,k-1}^2 \| b_{k-1}^* \|^2$) then
(10) exchange b_k and b_{k-1}, compute the Gram-Schmidt orthogonalization, $k=\max(k-1,2)$;
(11) else $k=k+1$;
(12) fi
(13) od
(14) return $b_1,b_2,\cdots,b_n \in Z^n$.

In algorithm 1, step (4) to step (8) is also called size-reduction. Suppose $M=\max(\| b_1 \|^2,\cdots,\| b_n \|^2)$, then the standard LLL algorithm needs about $O(n^5(\log M)^2)$ arithmetic operations for computing a LLL reduced lattice basis. With the fp-LLL algorithm (a fast sequential LLL algorithm), about 500 dimensional lattices could be reduced in the current personal computers.

III. ANALYSIS OF PARALLEL BLOCK-BASED LLL ALGORITHM

From the description of algorithm 1, it is difficult to implement it in parallel because of the continuous exchange and adjustment of two adjacent vectors. So people consider to split the basis into several blocks and parallelize them. In this section, we described and analyzed the traditional block-based LLL algorithm firstly, and then we proposed the new scheme and discussed it.

A. Traditional Block-based LLL Algorithm

There are at most about $O(n^2 \log M)$ swaps that are necessary to reduce a basis. The basic thought of parallelization is to complete as much swaps as possible at one time and the early all-swap LLL algorithm is based on this idea. The swaps are divided into two cases-odd and even cases. At one time, all the odd or even stations should be completed and it is a block-based parallel algorithm with block-size 2 in fact. For a lattice *L*, suppose the block-size be *l*, without loss of generality, we assume $n = ml$. It is to say that the basis $b_1,b_2,\cdots,b_n$ could be divided into *m* vector blocks, which of them has *l* vectors. The crucial idea of the block-based LLL algorithm is to calculate every block at one time and then combine them. Next we describe the block-based LLL algorithm with block-size *l* and blocks *m*.

Algorithm 2. Traditional block-based LLL algorithm

Input: Lattice basis $b_1,b_2,\cdots,b_n \in Z^n$, reduced parameter δ ($\frac{1}{4} < \delta \le 1$) and block-size *l*

Output: a LLL reduced with parameter δ

(1) compute the Gram-Schmidt orthogonalization;
(2) split the basis into *m* blocks of size *l*;
(3) repeat
(4) for($k=1; k \le m; k++$) do
(5) $LLL(b_{km+1}, b_{km+2}, \cdots, b_{km+l}, \delta)$;
(6) od
(8) for($k=1; k \le m; k++$) do
(9) compute and size-reduce of $\mu_{kl+1,kl}$;
(10) if($\delta \| b_{kl}^* \|^2 > \| b_{kl+1}^* \|^2 + \mu_{kl+1,kl}^2 \| b_{kl}^* \|^2$) then
(11) exchange b_{kl} and b_{kl+1};
(12) compute and size-reduce $\mu_{kl,i}$ and $\mu_{j,kl+1}$;
(13) fi
(14) for($k=1; k \le m; k++$) do
(15) update and size-reduce $\mu_{i,j}$;
(16) od
(17) od
(18) until there is no swap;
(19) return $b_1,b_2,\cdots,b_n \in Z^n$.

In algorithm 2, step 4 to step 6 could be done in parallel with *m* computing units. For the other steps, we must implement them in sequential. So algorithm 2 could be realized with master-slave architecture. However, it should be noted that there are two problems restricting the parallel

efficiency. Firstly, after every block being done, the borders of the blocks have to be checked for possible additional swaps. Also the $\mu_{kl,i}$ and $\mu_{j,kl+1}$ should be recomputed if the swap occurs. Then the parallel efficiency will be decreased severely. Secondly, if there is one swap, the algorithm should be implemented again. It is to say that the load of every computing unit is not balanced, which also leads to the decreasing of parallel efficiency.

For the second problem, because LLL algorithm should be judged and swapped continuously, the judges and swaps in the borders are not avoidable. So we cannot settle it unless we change the algorithm fundamentally. For the first problem, the judges and swaps could be done in parallel and the $\mu_{i,j}$ could be updated in the last. Then the operations in the borders could be implemented in parallel but the update in the last will be spent some extra operations. However, time of the update in the last is smaller than the operations in the borders. Based on this idea, we give a new scheme of parallel LLL algorithm.

B. New Scheme of Parallel LLL Algorithm

From algorithm 2, the sequential implementation in the borders limits the parallel efficiency. In fact, the $\mu_{i,j}$ could be updated in the last uniformly. Then we proposed the new scheme as follows.

Algorithm 3. New scheme of parallel LLL algorithm

Input: Lattice basis $b_1, b_2, \cdots, b_n \in Z^n$, reduced parameter δ ($\frac{1}{4} < \delta \le 1$) and block-size l

Output: a LLL reduced with parameter δ

There are m+1 computing units and m is the blocks in default. $p_1, p_2, \cdots, p_m$ are computing nodes, and p_0 is control node, determining algorithm whether to terminate.

(1) p_0 : compute the Gram-Schmidt orthogonalization;
(2) p_0 : split the basis into m blocks of size l;
(3) $send(b_{(k-1)l+1}, \cdots, b_{kl}, swap\_k; p_k (1 \le k \le m))$;
(4) each node computes as follows:
(5) $p_k (1 \le k \le m)$:
(6) if($swap\_k == 1$) then
(7) $LLL(b_{km+1}, b_{km+2}, \cdots, b_{km+l}, \delta)$;
(8) compute and size-reduce $\mu_{kl+1,kl}$;
(10) if($\delta \| b_{kl}^* \|^2 > \| b_{kl+1}^* \|^2 + \mu^2_{kl+1,kl} \| b_{kl}^* \|^2$) then
(11) exchange b_{kl} and b_{kl+1} ;
(12) $swap\_k = 1$;
(13) fi
(14) $send(b_{(k-1)l+1}, \cdots, b_{kl}, swap\_k; p_0)$;
(15) fi
(16) end
(17) p_0 :
(18) $recv(b_{(k-1)l+1}, \cdots, b_{kl}, swap\_k; p_k)$;
(19) if($sum(swap\_k) == 0$) then end;
(20) else
(21) compute the Gram-Schmidt orthogonalization;
(22) goto step 3;
(23) fi
(24) end
(25) return $b_1, b_2, \cdots, b_n \in Z^n$.

In algorithm 3, the operations will be implemented in parallel and the parallel efficiency is increased obviously. The Gram-Schmidt orthogonalization in the last guarantees that the $\mu_{i,j}$ could be computed. So the output of algorithm 3 is a LLL reduced basis. From the flow of algorithm 3, there are some extra operations in the last. However, the more time-consumed operations could be divided into the m computing units and done in parallel, which will decrease the running time.

IV. EXPERIMENTAL RESULTS

Algorithm 3 can get a LLL reduced basis and also has a good speedup. In this section, we implemented it in Sunway parallel processing system and offered some experimental results.

There are 16 Intel Core2 DUO 2.4GHz CPUs with 4 cores and 96GB main memory in the Sunway system. We use Linux with MPI compiler as the operation system and c++ as the programming language. The data types and some functions are based on package NTL [11]. In order to reflect the real factor, we wrote the basic source code for LLL algorithm but did not use the fpLLL algorithm with high efficiency in NTL.

The experimental objects are random prime lattices. Every element of the basis is prime generated randomly. We generated some n = 30,40,50−dimensional lattices randomly and the data length is about 1000 bits. Firstly, we tested the lattices with original LLL algorithm. Then we tested them with the traditional parallel scheme and the new parallel scheme of LLL algorithm. The results are as in table 1.

TABLE I. THE TEST RESULTS

Dimension	Block-size	LLL algorithm in sequential (second)	LLL algorithm in parallel (second)	
			Traditional scheme	New scheme
30	5	71	130	64
30	10	71	156	70
40	5	371	662	221
40	10	371	725	247
50	5	922	1882	445
50	10	922	1731	412

From the above table, the efficiency of traditional parallel scheme of LLL algorithm is much lower compared to the original LLL algorithm in sequential and the speedup factors are between 1/2 to 1. With the new scheme, the parallel LLL algorithm has some advantage on the original one and the speedup factor is about 2. However, the advantage of the block-based algorithm in parallel is not obvious. The main reason is that the load of every computing unit is not balanced. Also the relevancy of the two adjacent vectors

decides that the swaps cannot be completely at one time. Therefore, in order to increase the parallel efficiency of LLL algorithm severely, we must improve the LLL algorithm in the foundation but not just use the idea of splitting the basis into several blocks.

At the same time, for one basis of lattice, the efficiency will be not same if the block-size is different. For example, the running time of block-size 5 is smaller than block-size 10 for 30-dimensional lattices. But for 50-dimensional lattices, the result is reversed. For the different dimensional lattices, how to choose the proper block-size to get the high parallel efficiency is an important issue for us.

V. CONCLUSIONS

Parallelization is an important way to improve the efficiency of algorithms, and we studied the parallel implementation technology of LLL algorithm in this paper. Based on the bottleneck analysis of the traditional block-based LLL algorithm, we proposed a new parallel scheme on the LLL algorithm. The new scheme could get a LLL reduced basis and has a good speedup. At last, the experimental results verified the validity of the scheme.

Although the new scheme has obvious advantage on the traditional scheme, the parallel efficiency of block-based LLL algorithm is not ideal, which limit the extension to high dimension of LLL algorithm severely. Therefore, how to improve lattice basis algorithms (not only LLL algorithm) fundamentally and design new algorithms suitable for the multiprocessor computer architecture becomes the research focus of this area in the next step.

ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China (Grant Nos.61003291).

REFERENCES

[1] A K Lenstra, H W Lenstra, L Lovasz, “Factoring polynomials with rational coefficients,” Mathematiche Annalen, 1982, 261(4):515-534.

[2] L M Adlman, “On breaking generalized knapsack public key cryptosystems,” In: Pro. 15th ACM Symp. On Theory of Computing. ACM press, 1983, 402-412.

[3] D Coppersmith, “Finding a small root of a univariate modular equation,” In: U. Maurer. Proceedings of EUROCRYPT 1996. Berlin: Springer-Verlag, 1996, LNCS 1070: 155-165.

[4] S Santanu. “Some results on cryptanalysis of RSA and factorization,” PhD thesis, Indian Statistical Institute, Kolkata, 2011.

[5] R S Kumar, C Narasimam, S P Setty, “Lattice based tools in cryptanalysis for public key cryptography,” International Journal of Network Security & Its Applications, 2012, 4(2): 155-162.

[6] C P Schnorr, “Average time fast SVP and CVP algorithms for low density lattices and the factorization of integers,” Technical Report, 2010, 1–18.

[7] C P Schnorr, “Factoring integers and computing discrete logarithm via Diophantine approximation,” In: Advances in Computional Complexity, AMS, 1993, 171-182.

[8] W Backes, S Wetzel, “A Parallel LLL using POSIX Threads,” Technical report, Dept. of Computer Science, Stevens Institute of Technology, 2009.

[9] Susanne Wetzel, “An efficient parallel block-reduction algorithm,” In: ANTS-III, LNCS 1423, Berlin: Springer-Verlag, 1998, 323-337.

[10] P Q Nguyen, B Valle, “The LLL Algorithm: Survey and Applications,” 1st edition, Berlin: Springer Publishing Company, 2009: 19-71.

[11] V Shoup, “Number theory library(NTL) for c++,” http://www.shoup.net/ntl/, 2010.