Fast Lattice Basis Reduction Suitable for Massive Parallelization and Its Application to the Shortest Vector Problem

Tadanori Teruya[1], Kenji Kashiwabara[2][1], and Goichiro Hanaoka[1]

[1] Information Technology Research Institute, National Institute of Advanced Industrial Science and Technology. AIST Tokyo Waterfront Bio-IT Research Building, 2-4-7 Aomi, Koto-ku, Tokyo, Japan.

[2] Department of General Systems Studies, University of Tokyo. 3-8-1 Komaba, Meguro-ku, Tokyo, Japan.

Abstract. The hardness of the shortest vector problem for lattices is a fundamental assumption underpinning the security of many lattice-based cryptosystems, and therefore, it is important to evaluate its difficulty. Here, recent advances in studying the hardness of problems in large-scale lattice computing have pointed to need to study the design and methodology for exploiting the performance of massive parallel computing environments. In this paper, we propose a lattice basis reduction algorithm suitable for massive parallelization. Our parallelization strategy is an extension of the Fukase–Kashiwabara algorithm (J. Information Processing, Vol. 23, No. 1, 2015). In our algorithm, given a lattice basis as input, variants of the lattice basis are generated, and then each process reduces its lattice basis; at this time, the processes cooperate and share auxiliary information with each other to accelerate lattice basis reduction. In addition, we propose a new strategy based on our evaluation function of a lattice basis in order to decrease the sum of squared lengths of orthogonal basis vectors. We applied our algorithm to problem instances from the SVP Challenge. We solved a 150-dimension problem instance in about 394 days by using large clusters, and we also solved problem instances of dimensions 134, 138, 140, 142, 144, 146, and 148. Since the previous world record is the problem of dimension 132, these results demonstrate the effectiveness of our proposal.

Keywords: lattice basis reduction, parallelization, shortest vector problem, SVP Challenge

1 Introduction

1.1 Background

Lattice-based cryptography is an attractive field of research, because it has produced useful cryptographic schemes, for example, public-key cryptosystems, functional encryption schemes, and fully homomorphic encryption schemes, etc. [4,26,31,44,24,1,2,27,28,7,29,10,11,8,9,12], and these systems are believed to resist quantum cryptanalysis. Their security has been proven under the computational hardness assumption, and this assumption is related to the difficulty in solving the shortest vector problem (SVP) on a lattice. Therefore, it is important to show evidence of and to estimate the hardness not only theoretically, but also practically.

There have been many studies reporting on algorithms to solve the SVP, and the typical approaches are lattice basis reduction and enumerating lattice vectors. Lattice basis reduction algorithms take a lattice basis as input and output another lattice basis for the same lattice whose basis vectors are relatively shorter than the input. Enumeration algorithms take a lattice basis as input and search for relatively shorter lattice vectors than the basis vectors of the input. To evaluate the computational hardness of SVP, it is important to improve the efficiency of algorithms and implement high-performance solvers; here too, there have been many studies [36,33,49,47,48,13,37,14,21,16,30,17,23,15,46,34,20,38,39,6]. However, thanks to recent advances in computing technologies, exploiting the performance of massively parallel computing environment, for example, a cluster consists of many large computers, has become the most important strategy with which to evaluate the security of lattice cryptosystems, and a methodology to solve the SVP is needed for it.

1.2 Our Contribution

In this paper, we propose a new lattice basis reduction algorithm suitable for parallelization. Our proposal consists of a parallelization strategy and a reduction strategy.

Our parallelization strategy enables many parallel processes to reduce the lattice basis cooperatively, so it can exploit the performance of parallel computing environments in practice. This strategy is based on an idea proposed by Fukase and Kashiwabara in [20]; however, the parallel scalability of their algorithm is unclear with regard to massive parallelization. To achieve parallel scalability, our strategy consists of two methods. The first method is a procedure of generating different lattice bases for many processes. Our experiments show that this method enables the processes to generate a huge number of lattice vectors and that it works well even if the processes take the same lattice basis. The second method is to use global shared storage to keep information on the cooperative lattice basis reduction; this method enables many processes to share auxiliary information for accelerating the lattice basis reduction with a small synchronization overhead.

Our lattice basis reduction strategy enables us to efficiently obtain lattice bases with a small sum of squared lengths of orthogonal basis vectors, which is related to finding relatively short lattice vectors. Experiments and analyses are given by Fukase and Kashiwabara in [20], as explained in Section 3.2, and Aono and Nguyen in [5]. Fukase and Kashiwabara in [20] proposed a strategy to reduce this sum, but it is complicated. To efficiently obtain a lattice basis that has a small sum of squared lengths, we propose a new strategy based on an evaluation function of lattice bases.

We applied our algorithm to problem instances in the SVP Challenge, which is hosted by Technische Universität Darmstadt [45]. We solved a problem instance of dimension 150 in about 394 days with four clusters, a world record. In addition, we solved problem instances of dimensions 134, 138, 140, 142, 144, 146, and 148.

1.3 Related Work

Kannan's enumeration (ENUM) algorithm [33] is an exponential-time algorithm that outputs the shortest or nearly shortest lattice vector of given lattice. ENUM and its variants are used as the main routine or subroutine to solve the shortest lattice vector problem (SVP), as explained in Section 2.3. Thus, studying the ENUM algorithm is an important research field, and several improvements have been proposed [16,30,17,23,38].

The Lenstra–Lenstra–Lovász (LLL) algorithm [36] is a lattice basis reduction algorithm which takes a lattice basis and a parameter δ and produces a δ-LLL reduced basis of the same lattice. The LLL algorithm is the starting point of many lattice basis reduction algorithms and is a polynomial-time algorithm. Since the first basis vector of the output basis is a relatively short lattice vector in the lattice in practice, the LLL algorithm is also used as the main routine or subroutine of SVP algorithms. The Block Korkin–Zolotarev (BKZ) algorithm [49] is a block-wise generalization of the LLL algorithm, and it uses the ENUM algorithm as a subroutine. The BKZ algorithm generates relatively shorter basis vectors than the LLL algorithm does. Studying LLL, BKZ, and their variants is a popular topic of research and several improvements and extensions have been reported [21,15,6,39].

Schnorr's random sampling reduction (RSR) algorithm [47,48] is a probabilistic lattice basis reduction algorithm. The RSR algorithm randomly generates lattice vectors. The simple sampling reduction (SSR) algorithm [13,37,14] is an extension of the RSR algorithm that analyzes the expected length of the lattice vectors to be generated. The authors of SSR also presented several theoretical analyses of the RSR and their algorithm.

Our algorithm is based on the Fukase–Kashiwabara (FK) algorithm [20]. The FK algorithm is an extension of the SSR algorithm. The authors of the FK algorithm refined the calculation for the expected length of lattice vectors to be generated and showed that the probability of finding short lattice vectors can be increased by reducing the sum of squared lengths of orthogonal basis vectors. A brief introduction to the FK algorithm is given in Section 3.2. Aono and Nguyen [5] presented a theoretical analysis of the RSR algorithm and its extensions. They also gave a probabilistic perspective and a comparison with [23].

With the aim of using parallel computation to solve the SVP, several researchers proposed parallelized algorithms [16,30,17,46,34]. In particular, parallelized ENUM algorithms on CPUs, GPUs, and FPGAs were investigated by [16], [30], and [17]. A parallelized SSR algorithm for GPUs was proposed in [46]. A design for an SVP solver that exploits large-scale computing environments was proposed by [34]. In [34], the authors integrated, tuned up, and parallel pruned the ENUM algorithm [23] into a multicore-CPU and GPU implementation [30], and they extended this implementation by using multiple CPUs and GPUs in the single program multiple data (SPMD) style [41].

Organization. Preliminaries are described in Section 2. We then give a brief introduction to sampling reduction and the previous studies [47,48,37,20] in Section 3. An overview of our parallelization strategy is presented in Section 4, and our proposal is explained in detail in Section 5. Section 6 presents the results of applying our proposal to the SVP Challenge and we conclude in Section 7.

2 Preliminaries

2.1 Lattice

In this section, we introduce notations and definitions of the lattice

The set of natural numbers $\{0, 1, \ldots\}$ is denoted by $\mathbb{N}$, and the sets of integers and real numbers are denoted by $\mathbb{Z}$ and $\mathbb{R}$, respectively. We denote an n-element vector (or sequence) as $\boldsymbol{x} = (x_1, \ldots, x_n) = (x_i)_{i=1}^n$. Let $\boldsymbol{v} = (v_1, \ldots, v_n) \in \mathbb{R}^n$ and $\boldsymbol{u} = (u_1, \ldots, u_n) \in \mathbb{R}^n$ be two n-dimensional vectors on $\mathbb{R}$; we define the inner product of $\boldsymbol{v}$ and $\boldsymbol{u}$ by $\langle \boldsymbol{v}, \boldsymbol{u} \rangle := \sum_{i=1}^n v_i u_i$. The length (Euclidean norm) of $\boldsymbol{v}$, denoted by $\|\boldsymbol{v}\|$, is $\sqrt{\langle \boldsymbol{v}, \boldsymbol{v} \rangle}$.

Let $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n \in \mathbb{Z}^n$ be n-dimensional linearly independent column (integral) vectors. The *(full rank) lattice L spanned by a matrix $\boldsymbol{B} = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$* is defined as the set $L = \mathcal{L}(\boldsymbol{B}) = \{\boldsymbol{B}\boldsymbol{x} \mid \boldsymbol{x} \in \mathbb{Z}^n\}$. Such a $\boldsymbol{B}$ is called a lattice basis of L, and each $\boldsymbol{v} \in L$ is called a lattice vector of L. Every lattice has infinitely many lattice bases except $n = 1$; i.e., let U be an n-dimensional unimodular matrix over $\mathbb{Z}$; then $\mathcal{L}(\boldsymbol{B}) = \mathcal{L}(\boldsymbol{B}U)$, so that $\boldsymbol{B}U$ is another basis of L if U is not the identity matrix. For a lattice basis $\boldsymbol{B} = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$, the corresponding *(Gram–Schmidt) orthogonal basis vectors* $\boldsymbol{b}_1^*, \ldots, \boldsymbol{b}_n^* \in \mathbb{R}^n$ are defined by $\boldsymbol{b}_i^* = \boldsymbol{b}_i - \sum_{j=1}^{i-1} \mu_{\boldsymbol{B},i,j} \boldsymbol{b}_j^*$, where $\mu_{\boldsymbol{B},i,j} = \langle \boldsymbol{b}_i, \boldsymbol{b}_j^* \rangle / \|\boldsymbol{b}_j^*\|^2$. The *determinant of* L is denoted by $\det(L)$, and note that $\det(L) = \prod_{i=1}^n \|\boldsymbol{b}_i^*\|$ for any basis $\boldsymbol{B} = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$ of L.

Given a lattice basis $\boldsymbol{B} = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$ of a lattice L, the *i-th orthogonal projection by $\boldsymbol{B}$*, denoted by $\pi_{\boldsymbol{B},i} : \mathbb{R}^n \longrightarrow \operatorname{span}(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{i-1})^\perp$, is defined as $\pi_{\boldsymbol{B},i}(\boldsymbol{v}) = \boldsymbol{v} - \sum_{j=1}^{i-1} v_j^* \boldsymbol{b}_j^*$, where $v_j^* = \langle \boldsymbol{v}, \boldsymbol{b}_j^* \rangle / \|\boldsymbol{b}_j^*\|^2 \in \mathbb{R}$ (for $i = 1$, $\pi_{\boldsymbol{B},1}$ is the same as the identity map). Note that $\|\pi_{\boldsymbol{B},i}(\boldsymbol{v})\|^2 = \sum_{j=i+1}^n \left(\langle \boldsymbol{v}, \boldsymbol{b}_j^* \rangle / \|\boldsymbol{b}_j^*\| \right)^2$. We define the *$i$-th orthogonal complement by $\boldsymbol{B}$* as $L_{\boldsymbol{B},i} := \pi_{\boldsymbol{B},i}(L) = \{\pi_{\boldsymbol{B},i}(\boldsymbol{v}) \mid \boldsymbol{v} \in L\}$. We call $\|\pi_{\boldsymbol{B},i}(\boldsymbol{v})\|$ the *orthogonal length of $\boldsymbol{v}$ on $L_{\boldsymbol{B},i}$*; this is an important measurement in the context of lattice basis reduction.

2.2 Natural Number Representation

In this section, we explain the natural number representation (NNR) of a lattice vector. In Schnorr's algorithm [47], a lattice vector is represented by an n-dimensional sequence consisting of 0s and 1s, where n is the dimension of the given lattice basis. The numbers 0 and 1 express the amount of aberration from the orthogonal point at each index, and one can calculate a lattice vector from the lattice basis and this sequence. Moreover, Buchmann and Ludwig [13,37,14] indicate that one can generalize 0, 1 to the natural numbers 0, 1, 2, . . ., and calculate the expected value of squared lengths of the lattice vectors generated from a sequence of natural numbers. Fukase and Kashiwabara [20] call such a sequence of natural numbers a *natural number representation (NNR)*, and Aono and Nguyen [5] call it a *tag (defined on the natural numbers)*. One can determine a set of NNRs that has a small expected value of squared lengths for a given basis. The following definitions and statements are basically reprinted from [20], but similar descriptions are given in other papers [13,37,14,5].

Definition 1 (Natural Number Representation). *Let $\boldsymbol{B} = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$ be a lattice basis of the lattice L. Given a lattice vector $\boldsymbol{v} = \sum_{i=1}^n \nu_i \boldsymbol{b}_i^* \in L$, the* natural number representation (NNR) of $\boldsymbol{v}$ on $\boldsymbol{B}$ *is a vector of n non-negative integers* $(\mathfrak{n}_1, \ldots, \mathfrak{n}_n) \in \mathbb{N}^n$ *such that* $-(\mathfrak{n}_i + 1)/2 < v_i \leq -\mathfrak{n}_i/2$ *or* $\mathfrak{n}_i/2 < v_i \leq (\mathfrak{n}_i + 1)/2$ *for all* $i = 1, \ldots, n$.

Theorem 1 ([20]). *Let* $\boldsymbol{B} = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$ *be a lattice basis of the lattice L, and let* $\boldsymbol{v} = \sum_{i=1}^{n} v_i \boldsymbol{b}_i^* \in L$. *For any lattice vector* $\boldsymbol{v} \in L$, *the NNR of* $\boldsymbol{v}$ *is uniquely determined, and the map* $\mathbf{n}_{\boldsymbol{B}} : L \to \mathbb{N}^n$, *which is given by* $\mathbf{n}_{\boldsymbol{B}}(\boldsymbol{v}) := (\mathfrak{n}_1, \ldots, \mathfrak{n}_n)$, *is a bijection.*

Assumption 1 (Randomness Assumption [47,20,5]) *Let* $\boldsymbol{v} = \sum_{j=1}^{n} v_j \boldsymbol{b}_j^*$ *be a lattice vector, and* $\boldsymbol{v}$ *has the NNR* $\mathfrak{n} = (\mathfrak{n}_1, \ldots, \mathfrak{n}_n)$. *The coefficients* v_j *of* $\boldsymbol{v}$ *are uniformly distributed in* $(-(\mathfrak{n}_j + 1)/2, -\mathfrak{n}_j/2]$ *and* $(\mathfrak{n}_j/2, (\mathfrak{n}_j + 1)/2]$ *and statistically independent with respect to j.*

The difference from the original randomness assumption [47,48] is that the NNRs are generated by a procedure called the sampling algorithm and these NNRs belong to a subset of $\{0, 1\}^n$ defined as

$$\{(0, \ldots, 0, \mathfrak{n}_{n-u}, \ldots, \mathfrak{n}_{n-1}, 1) \mid \mathfrak{n}_{n-u}, \ldots, \mathfrak{n}_{n-1} \in \{0, 1\}\},$$

where u is a positive integer parameter of the sampling algorithm.

Theorem 2 ([20]). *Let* $\boldsymbol{B} = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$ *be a lattice basis of the lattice L, and let* $\mathfrak{n} = (\mathfrak{n}_1, \ldots, \mathfrak{n}_n)$ *be an NNR. The expectation of the squared length of the corresponding lattice vector* $\boldsymbol{v}$ *of* $\mathfrak{n}$ *(namely,* $\mathbf{n}_{\boldsymbol{B}}(\boldsymbol{v}) = (\mathfrak{n}_1, \ldots, \mathfrak{n}_n)$*) is:*

$$E\left[\|\boldsymbol{v}\|^2\right] = \frac{1}{12} \sum_{i=1}^{n} (3\mathfrak{n}_i^2 + 3\mathfrak{n}_i + 1) \left\|\boldsymbol{b}_i^*\right\|^2. \tag{1}$$

Theorem 2 states that one can calculate the expected value of the squared length of a generated lattice vector with given NNR in a given basis without calculating the coordinates of the lattice vector. For a relatively reduced basis (the output of the LLL algorithm, the BKZ algorithm, or other variants), the sequence $(\|\boldsymbol{b}_1^*\|^2, \|\boldsymbol{b}_2^*\|^2, \ldots, \|\boldsymbol{b}_n^*\|^2)$ tends to decrease with respect to the indices. Therefore, a short lattice vector tends to have natural number 0 in first consecutive indices of the NNR, and natural numbers 1 and 2 appear in last consecutive indices in the NNR for any relatively reduced basis.

2.3 Shortest Vector Problem and SVP Challenge

Given a lattice basis $\boldsymbol{B}$ of a lattice L of dimension n, the *shortest vector problem (SVP)* consists in finding the shortest non-zero vector $\boldsymbol{v} \in L$. The SVP is NP-hard under randomized reduction [3]. It is hard to find the shortest lattice vector in the lattice when n is large. However, thanks to the *Gaussian heuristic* $\mathrm{GH}(L)$, the length of the shortest lattice vector in L can be estimated as $\mathrm{GH}(L) := \left(\Gamma(n/2 + 1) \cdot \det(L)\right)^{1/n}/\sqrt{\pi}$, where Γ is the gamma function [32,40]. The *root Hermite factor of* $\boldsymbol{v}$ *and L* is defined by $\left(\|\boldsymbol{v}\| / \det(L)^{1/n}\right)^{1/n}$. Given a lattice vector $\boldsymbol{v}$, the corresponding root Hermite factor is used to make comparative measurements among different lattices, because it is independent of the lattice basis [22,42].

The security of lattice-based cryptosystems depends on the hardness of the lattice problem. Therefore, it is important to study efficient algorithms for solving the SVP. Here, a competition, called the *SVP Challenge* [45], was maintained by Technische Universität Darmstadt since 2010. The SVP Challenge gives a problem instance generator that produces SVP instances and hosts a ranking of registered lattice vectors as follows. A submitted lattice vector $\boldsymbol{v} \in L$ is registered if one of the following conditions is satisfied: no lattice vector is registered at the dimension of $\boldsymbol{v}$ and $\|\boldsymbol{v}\| < 1.05 \cdot \mathrm{GH}(L)$, or $\boldsymbol{v}$ is shorter than any other lattice vector of the same dimension. In this paper, for a lattice L, we call a lattice vector $\boldsymbol{v} \in L$ a *solution of* L if $\|\boldsymbol{v}\| < 1.05 \cdot \mathrm{GH}(L)$.

3 Brief Introduction to Sampling Reduction

Before explaining our algorithm, we briefly introduce the RSR algorithm and its extensions [47,48,37,20].

3.1 Overview of Sampling Reduction

Here, we briefly introduce the RSR algorithm and its extensions [47,48,37,20] and describe the parts that our algorithm shares with them.

First of all, we define the following notions and terminology.

Definition 2 (lexicographical order). *Let* $\boldsymbol{B} = (\boldsymbol{b}_1, \dots, \boldsymbol{b}_n)$ *and* $\boldsymbol{C} = (\boldsymbol{c}_1, \dots, \boldsymbol{c}_n)$ *be two lattice bases of the same lattice. We say that* $\boldsymbol{B}$ is smaller than $\boldsymbol{C}$ in the lexicographical order *if and only if* $\|\boldsymbol{b}_j^*\| < \|\boldsymbol{c}_j^*\|$, *where* j *is the smallest index for which* $\|\boldsymbol{b}_j^*\|$ *and* $\|\boldsymbol{c}_j^*\|$ *are different.*

Definition 3 (insertion index). *Given a lattice vector* $\boldsymbol{v}$ *and a real number* δ *with* $0 < \delta \leq 1$, *the* δ-insertion index of $\boldsymbol{v}$ for $\boldsymbol{B}$ *is defined by*

$$\mathbf{h}_\delta(\boldsymbol{v}) = \min\left(\{i \mid \|\pi_{\boldsymbol{B},i}(\boldsymbol{v})\|^2 < \delta\|\boldsymbol{b}_i^*\|^2\} \cup \{\infty\}\right).$$

Note that the δ-insertion index is a parameterized notion of Schnorr [47,48].

The RSR algorithm and its extensions need a lattice reduction algorithm to reduce a given lattice basis in lexicographical order, and the LLL and BKZ algorithms can be used for this purpose. In this paper, we define this operation as follows.

Definition 4. *Let* $\boldsymbol{v} \in L$ *be a lattice vector with* $\|\pi_{\boldsymbol{B},i}(\boldsymbol{v})\| < \delta\|\boldsymbol{b}_i^*\|$. *We call the following operation* δ-LLL reduction of $\boldsymbol{B}$ at index i by $\boldsymbol{v}$: *generate an* $(n+1) \times n$ *matrix* $M = (\boldsymbol{b}_1, \dots, \boldsymbol{b}_{i-1}, \boldsymbol{v}, \boldsymbol{b}_i, \dots, \boldsymbol{b}_n)$, *compute the* δ*-LLL reduced (full rank) lattice basis* $\boldsymbol{B}'$ *by using the LLL algorithm with input* M, *and output the resulting lattice basis.*

As mentioned above, one can use another lattice reduction algorithm. In fact, the RSR algorithm [47,48] uses the BKZ algorithm.

The RSR algorithm and its extensions consist of two major parts. The first part takes a lattice basis $\boldsymbol{B}$ and a set of NNRs as input and generates a lattice vector that corresponds to an NNR and $\boldsymbol{B}$. It outputs a set of generated lattice vectors. In this paper, we call this operation *lattice vector generation*. The second part performs lattice basis reduction. If the result of the lattice vector generation is a lattice vector $\boldsymbol{v}$ whose orthogonal length is shorter than a orthogonal basis vector of $\boldsymbol{B}$, in other words, whose δ-insertion index is finite, $\boldsymbol{B}$ is reduced by using δ-LLL reduction of $\boldsymbol{B}$ at index $\mathbf{h}_\delta(\boldsymbol{v})$ by $\boldsymbol{v}$. Note that the lexicographical order of the resulting lattice basis of this part is smaller than the input. To summarize the RSR algorithm and its extensions, alternately repeat the lattice vector generation and the lattice basis reduction to reduce a given lattice basis. To solve the SVP, these calculations are repeated until the length of the first basis vector of the obtained lattice basis is less than $1.05 \cdot \mathrm{GH}\left(\mathcal{L}(\boldsymbol{B})\right)$, where $\boldsymbol{B}$ is the given lattice basis.

To reduce the lattice basis or solve the SVP efficiently, it is important to choose a lattice vector as input for the lattice basis reduction from the result of the lattice vector generation. Each previous study has its own selection rule. In the RSR algorithm, a lattice vector $\boldsymbol{v}$ with $\mathbf{h}_\delta(\boldsymbol{v}) \leq 10$ is chosen from the result of the lattice vector generation. In the SSR algorithm, a lattice vector $\boldsymbol{v}$ with $\mathbf{h}_\delta(\boldsymbol{v}) = 1$ is chosen. Fukase and Kashiwabara [20] introduce a rule to choose a lattice vector, called restricting reduction, as explained in Section 3.2.

Ludwig [37] and Fukase and Kashiwabara [20] computed the expectation of the squared length of the generated lattice vectors in the lattice vector generation. According to the expectation value of squared length, their algorithm prepared lists of NNRs with 0, 1, and 2, and generated the corresponding lattice vectors from the lattice basis. Aono and Nguyen [5] presented an algorithm to produce a set of better NNRs (in their terms, tags).

3.2 Fukase–Kashiwabara Algorithm

As our algorithm is an extension of a parallelization strategy and uses the global shared storage proposed by Fukase and Kashiwabara [20], we will briefly introduce their proposal before explaining ours. Note that our algorithm does not use the restricting reduction of their algorithm. We refer the reader to [20] for the details of the FK algorithm.

Restricting Reduction. Ludwig [37] and Fukase and Kashiwabara [20] showed that one can calculate the expected value of the squared lengths of the generated lattice vectors for a lattice basis and a set of NNRs, and this expectation is proportional to the sum of the squared lengths of the orthogonal basis vectors. Therefore, a nice reduction strategy seems to reduce the lattice basis so as to decrease the sum of squared lengths, and hence, such a strategy is considerable. Fukase and Kashiwabara [20] proposed a method for decreasing the sum of squared lengths by introducing the restricting reduction index; this strategy is briefly explained below.

To decrease the sum of squared lengths quickly, some application of a lattice vector to the lattice basis is not good. Moreover, some application of a lattice vector which decreases the sum of squared lengths is not the fast way to reach a very small sum of squared lengths. To decrease the sum of squared lengths quickly, Fukase and Kashiwabara [20] used a restricting index. For an index ℓ, called the restricting reduction index, the lattice basis is reduced at an index ℓ' only after ℓ. In other words, the basis vectors before ℓ are fixed. Let ℓ be 0 at the beginning of the program. As the main loops are executed, the restricting index is gradually increased according to some rule. When the restricting index reaches a certain value, it is set to 0 again. By using the restricting index, the sum of squared lengths of orthogonal basis vectors is decreased from first consecutive indices.

Stock Vectors. Some lattice vectors may be useful even if they are not short enough for reducing the lattice basis. For example, after reduction of the lattice basis at index i by another lattice vector, a lattice vector $\boldsymbol{v}$ may be applied at index $i + 1$ or greater. In addition, after changing the restricting index to 0, a lattice vector $\boldsymbol{v}$ may be applied to the lattice basis if it was not used for reducing the lattice basis because of a previous high restriction index. Therefore, lattice vectors that are relatively shorter than the i-th orthogonal basis vector are stored in global shared storage, and each stored lattice vector is associated with an orthogonal complement $L_{\boldsymbol{B},i}$ (namely, the place of the stored vector is determined by the basis vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{i-1}$) in order to load and use it for reducing the lattice basis by all the parallel running processes. Fukase and Kashiwabara [20] call these stored vectors *stock vectors*.

Parallelization. A set of generated lattice vectors is determined from the lattice basis and set of NNRs. Therefore, even if the sets of NNRs are the same, two different lattice bases generate different sets of lattice vectors that have few vectors in common. Fukase and Kashiwabara [20] utilized this heuristic approach with stock vectors to generate a lot of lattice vectors and cooperate with each parallel running process. Given a lattice basis, their algorithm generates different lattice bases except for first consecutive indices from 1 and distributes each generated basis to each process. In other words, the processes have lattice bases that have different basis vectors at last consecutive indices. Since these lattice bases have a common orthogonal complement at first consecutive indices, processes can share stock vectors associated with common basis vectors of first consecutive indices, and since they have different basis vectors at last consecutive indices, the generated lattice vectors could be different from each other.

Note that basis reduction is done by each process, and keeping first consecutive basis vectors the same is difficult. Thus, some mechanism is required to keep first consecutive basis vectors the same among all the processes. Fukase and Kashiwabara [20] proposed a method by storing basis vectors as the stock vectors.

4 Overview of Our Parallelization Strategy

First, we briefly explain parallelization and the problems that make it hard to exploit the performance of parallel computation. Then, we briefly describe our strategy.

4.1 Technical Hurdles and Brief Review of Parallelization Methodology

Several researchers have used parallel computation to solve the SVP. In particular, there are three types of parallel algorithm for solving it.

The first type is the divide-and-conquer approach for parallelizing the ENUM, pruned ENUM, and sampling algorithms of lattice vectors [16,30,17,23,34,46]. Several researchers [16,30,17,34] studied parallel ENUM algorithms based on this methodology. This type of parallelization can be used to speed up the search for the shortest or a relatively short lattice vector. However, it is hard to solve high-dimensional SVPs by using this type of parallelization, which take an exponentially long time in practice. Hence, this type of parallelization is out of the scope of this paper.

The second type is randomization of lattice bases to exploit the power of massively parallel computing environments. The authors of reference [34] proposed an implementation to solve high-dimensional SVPs by exploiting massively parallel running processes. Their implementation works as follows. For a given lattice basis, the method generates a lot of lattice bases by multiplications with random unimodular matrices. The generated lattice bases are distributed to the processes, and each process runs the BKZ and pruned parallel ENUM algorithms. This method reduces a lot of lattice bases simultaneously, so the probability of finding a short lattice vector becomes higher. However, each process works independently; therefore, the information for accelerating lattice basis reduction is not shared among processes.

The third type of parallelization, proposed by Fukase and Kashiwabara [20], involves alteration of last consecutive basis vectors, which is briefly explained in Section 3.2. For a given lattice basis, their method generates a lot of lattice vectors from the given basis and a set of NNRs in order to find lattice vectors with a shorter length than the given orthogonal basis vectors. Their key idea consists of two heuristic approaches. Firstly, each process executes lattice vector generation, which takes the same set of NNRs and a different lattice basis as input in order to generate a different set of lattice vectors. To generate many different lattice bases, each process computes the δ-LLL reduction of a lattice basis of dimension n at last consecutive indices $\ell, \ell+1, \ldots, n$ by using a different lattice vector. Note that different lattice bases might be outputted for different input lattice vectors. Secondly, if the processes have different lattice bases but same first consecutive basis vectors with indices $1, 2, \ldots, \ell'$, then useful lattice vectors, which are contained in the same orthogonal complements (called stock vectors by Fukase and Kashiwabara), could be shared among the processes in order to accelerate the lattice basis reduction. Fukase and Kashiwabara proposed a parallelization strategy based on these heuristics. However, they reported experimental results with only 12 or fewer parallel processes. In fact, in our preliminary experiments of the parallelization strategy of Fukase and Kashiwabara [20], we observed that many parallel running processes to solve higher dimensional problems did not keep the same first consecutive basis vectors. Since current massively parallel computing environments have many more physical processing units (often, more than 100), we conclude that there is still room for scalable massive parallelization.

The parallelization strategy in our algorithm is an extension of [20]. The key idea of our method is explained next.

4.2 Key Idea of Our Parallelization Strategy

To reduce a lattice basis by utilizing the performance of a parallel computing environment effectively, one needs a scalable parallelization strategy that keeps first consecutive indices' basis vectors common but last consecutive indices' basis vectors different on the many lattice bases of the individual processes with a small synchronization penalty in practice. To accomplish this, we extend the parallelization strategy of Fukase and Kashiwabara [20]. Our strategy consists of two methodologies. The first enables each process to generate a lattice basis variant which has different last consecutive indices' basis vectors from those of other processes, and the second enables each process to share first consecutive indices' basis vectors. The rest of this section briefly explains our idea, and Section 5.2 explains it in detail.

Our lattice basis reduction with process-unique information (e.g., process ID, MPI rank, etc.) generates many lattice basis variants that have different last consecutive indices' basis vectors. In addition, each process generates a lot of lattice vectors to reduce the lattice basis by using a given lattice basis and a set of NNRs. If there is a lattice vector whose orthogonal length is relatively shorter than the orthogonal basis vectors in first consecutive indices, the process stores it in the global shared storage of stock vectors, by using the method proposed by Fukase and Kashiwabara [20], as explained in Section 3.2. Thanks to the process-unique information, each process can use this information as a seed for generating different lattice bases even if each process takes the same lattice basis as input. During the reduction execution, the processes work independently except for the access to the global shared storage and do not communicate with each other. The most computationally expensive operation is generating a lot of lattice vectors, and this operation is calculated in each process independently. Hence, there is only a negligible synchronization penalty in practice.

As mentioned above regarding the second method, Fukase and Kashiwabara [20] suggested that cooperation among parallel running processes can be enabled by sharing first consecutive indices' basis vectors. However, maintaining this situation without incurring a painfully large synchronization cost is a difficult task when there are many processes. Fukase and Kashiwabara [20] proposed to use global shared storage for stock vectors, as explained in Section 3.2. However, it is unclear how their method can be used to keep common basis vectors in first consecutive indices among many processes. We propose a method to share the basis vectors in first consecutive indices by using additional global shared storage. In our algorithm, each process has its own lattice basis. In every loop, each process stores basis vectors in first consecutive indices of its own lattice basis in this extra global shared storage; then it loads lattice vectors from the storage and computes the smallest lattice basis in lexicographical order by using the stored lattice vectors and their own lattice bases. Note that only first consecutive indices' basis vectors are stored in the global storage. In this method, many processes cause accesses to the global shared storage, and synchronization is required. However, the synchronization penalty is practically smaller than that of communication among many processes, and as mentioned above, the most computationally expensive operation is generating lattice vectors.

5 Our Algorithm

The most important part of our proposal is the methodology of parallelization. We designed and implemented our parallelization methodology in the single program multiple

```
Input: A lattice basis $\boldsymbol{B}$, process-unique information pui (regarded as an integer), two
       sets $S$ and $S'$ of natural number representations, integers $\ell_{\mathrm{fc}}$, $\ell_{\mathrm{lc}}$, and $\ell_{\mathrm{link}}$, and
       real values $\delta_{\mathrm{stock}}$, $\delta$, $\delta'$, $\delta''$, and $\Theta$.
Output: A lattice basis $\boldsymbol{B}$.
1  begin
2      loop begin
3          $\boldsymbol{B}' \leftarrow \boldsymbol{B}$;
4          Reduce $\boldsymbol{B}$ by using Algorithm 2 with input parameters $\boldsymbol{B}$, pui, $S'$, $\ell_{\mathrm{fc}}$, $\ell_{\mathrm{lc}}$,
             $\delta_{\mathrm{stock}}$, $\delta$, $\delta'$ and $\delta''$ (explained in Section 5.2);
5          Generate a set of lattice vectors $V$ from $\boldsymbol{B}$ and a set $S$ of natural number
             representations;
6          foreach $\boldsymbol{v} \in V$ do
7              if there exists an index $i = \mathbf{h}_{\delta_{\mathrm{stock}}}(\boldsymbol{v})$ such that $1 \leq i \leq \ell_{\mathrm{fc}}$ then store $\boldsymbol{v}$ in the
                 global shared storage of stock vectors (explained in Section 3.2);
8          end
9          Reduce $\boldsymbol{B}$ by using our strategy with inputs lattice vectors from the global
             shared storage of stock vectors and the parameter $\Theta$ (explained in Section 5.1);
10         Store the basis vectors of indices from 1 to $\ell_{\mathrm{link}}$ of $\boldsymbol{B}$ in the global shared
             storage of link vectors (explained in Section 5.2);
11         Load the lattice vectors from the global shared storage of link vectors; then
             generate the smallest lattice basis $\boldsymbol{B}''$ in lexicographical order by using loaded
             link vectors and $\boldsymbol{B}$; then replace $\boldsymbol{B}$ by $\boldsymbol{B}''$ (explained in Section 5.2);
12         if there exists a lattice vector $\boldsymbol{v}$ in the global shared storage of stock vectors and
             $\boldsymbol{B}$ with $\|\boldsymbol{v}\| < 1.05 \cdot \mathrm{GH}\left(\mathcal{L}(\boldsymbol{B})\right)$ then output $\boldsymbol{v}$ and halt this process;
13         if $\boldsymbol{B}' = \boldsymbol{B}$ then halt this process and output $\boldsymbol{B}$;
14     end
15 end
```

Algorithm 1: Main routine of our algorithm

```
Input: A lattice basis $\boldsymbol{B} = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$, the process-unique information pui (regarded
       as an integer), a set $S'$ of natural number representations, integers $\ell_{\text{fc}}$, $\ell_{\text{lc}}$, and
       real values $\delta_{\text{stock}}$, $\delta$, $\delta'$, and $\delta''$.
Output: A lattice basis $\boldsymbol{B}$.
1  begin
2      $\delta'_i \leftarrow \delta'$ for $i = \ell_{\text{lc}} + 1, \ldots, n$ where $n$ is the dimension of given lattice;
3      loop begin
4          Generate a set of lattice vectors $V$ from $\boldsymbol{B}$ and a set $S'$ of natural number
             representations;
5          foreach $\boldsymbol{v} \in V$ do
6              if there exists an index $i = \mathbf{h}_{\delta_{\text{stock}}}(\boldsymbol{v})$ such that $1 \leq i \leq \ell_{\text{fc}}$ then store $\boldsymbol{v}$ in the
                 global shared storage of the stock vectors (explained in Section 3.2);
7          end
8          Collect the lattice vectors $V' = \{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_N\}$ that have a $\delta$-insertion index at an
             index $i$ with $\ell_{\text{lc}} < i$ from $V$;
9          $\boldsymbol{B}' \leftarrow \boldsymbol{B}$;
10         loop begin
11             Find the lattice vector $\boldsymbol{v}_i \in V'$ with $j = \mathbf{h}_{\delta'_j}(\boldsymbol{v})$ such that $\ell_{\text{lc}} < j \leq n$ and
                 $\|\boldsymbol{b}^*_j\|^2 - \|\pi_{\boldsymbol{B},j}(\boldsymbol{v}_i)\|^2$ is maximum;
12             if $\boldsymbol{v}_i$ is not found in line 11 then go to line 18;
13             Reduce $\boldsymbol{B}$ by $\delta$-LLL reduction at the index $j$ by $\boldsymbol{v}_i$;
14             foreach $k = \ell_{\text{lc}} + 1, \ldots, j - 1$ do $\delta'_k \leftarrow \delta'_k - \delta''$;
15             foreach $k = j, \ldots, n$ do $\delta'_k \leftarrow \delta'$;
16             if $(i + \texttt{pui}) \bmod N = 0$ then go to line 18;
17         end
18         if $\boldsymbol{B} = \boldsymbol{B}'$ then terminate this subroutine and output $\boldsymbol{B}$;
19     end
20 end
```

Algorithm 2: Subroutine of our algorithm to generate lattice basis variants

data (SPMD) style [41]. Namely, each process executes the same program, but each process has a different internal state from each other. Our parallelization is an extension of the parallelization strategy and uses the global shared storage and stock vectors proposed by Fukase and Kashiwabara [20], as explained in Section 3.2. In addition, it contains a lattice basis reduction strategy based on our new evaluation function.

Our algorithm, listed in Algorithms 1 and 2, is based on the Fukase–Kashiwabara (FK) algorithm [20]; in particular, its lattice basis reduction is similar to that method, namely, by generating a lot of lattice vectors by using a lattice basis $\boldsymbol{B}$ and a set of natural number representations (NNRs), as defined by Definition 1 in Section 2.2, and reducing $\boldsymbol{B}$ to significantly decrease the sum of squared lengths of orthogonal basis vectors of $\boldsymbol{B}$ by using the generated vectors.

Our method to generate a lot of lattice vectors from a lattice basis is similar to the FK algorithm. Note that Aono and Nguyen [5] presented an algorithm to produce a set of better NNRs (in their terms, tags), but we did not use it.

The major differences from the FK algorithm are as follows: a parallelization strategy, and a lattice basis reduction strategy based on our evaluation function in order to decrease the sum of squared lengths significantly for each process. In Section 5.1, we show our new lattice basis reduction strategy for each process. In Section 5.2, we describe our new parallelization strategy. We explain how to choose parameters of our proposed algorithm in Section 5.3.

5.1 Basis Reduction Strategy using Evaluation Function

As mentioned above, our algorithm is based on the FK algorithm [20]. FK algorithm adopted restricting index as its lattice basis reduction method. But our algorithm uses a following evaluation function of bases on lattice basis reduction. Our algorithm generates a lot of lattice vectors by using a lattice basis $\boldsymbol{B}$ and a set of NNRs. Discovered relatively shorter lattice vectors, called stock vectors, are stored in global shared storage [20], as explained in Section 3.2. After generating these stock vectors, our algorithm uses them to reduce $\boldsymbol{B}$. Several stock vectors have the δ-insertion index at an index i for $\boldsymbol{B}$, which is defined in Section 2.1, so that there are several choices of stock vector that can reduce $\boldsymbol{B}$. Now we face the problem of deciding which stock vector is the best to reduce $\boldsymbol{B}$. An answer to this question is important for finding a solution efficiently.

However, based on our preliminary experiments with reducing higher dimensional lattice bases, this is quite difficult problem. In order to manage to resolve this difficulty, we introduce the following evaluation function Eval and a positive real number parameter Θ less than 1 for the lattice basis $\boldsymbol{B} = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$:

$$\mathsf{Eval}(\boldsymbol{B}, \Theta) = \sum_{i=1}^{n} \Theta^i \|\boldsymbol{b}_i^*\|^2. \tag{2}$$

It uses the following strategy at line 9. Replace $\boldsymbol{B}$ by a lattice basis $\overline{\boldsymbol{B}}$ which has the minimum $\mathsf{Eval}(\overline{\boldsymbol{B}}, \Theta)$, where $\overline{\boldsymbol{B}}$ is computed by the δ-LLL reduction of $\boldsymbol{B}$ at index i by $\boldsymbol{v}$, which is a lattice vector from the global shared storage of stock vectors with $\mathbf{h}_{\delta_{\text{stock}}}(\boldsymbol{v}) = i \leq \ell_{\text{fc}}$.

Now let us precisely describe what is difficulty and why we introduced Eval and Θ. According to [13,37,14,20], the average squared lengths of lattice vectors generated by $\boldsymbol{B}$ and the set of NNRs are proportional to the sum of squared lengths of orthogonal basis vectors of $\boldsymbol{B}$, explained in Section 2.2. In this way, the SVP can be solved by searching for shorter lattice vectors and using the found lattice vector to reduce the lattice basis. At this time, lattice basis reduction with a shorter lattice vector yields a reduced lattice basis in the context of the lexicographical order, which is defined in Section 2.1; however, the sum of the squared lengths of orthogonal basis vectors of the resulting lattice basis becomes much larger in practice. Namely, there is a trade-off between two reduction strategies (decreasing the lexicographical order and the sum of squared lengths of orthogonal basis vectors). To resolve this dilemma, a method for optimally choosing between these strategies is needed. However, as mentioned above, it seems to be quite difficult because, to the best of our knowledge, basic theories allowing this have not been developed in this research area. In order to manage this issue, we decide to combine both strategies and this is the reason why we introduced Eval and Θ.

The evaluation function Eval attempts to combine the lexicographical order and the order defined by the sum of squared lengths with an adjustable parameter Θ. For a lattice basis $\boldsymbol{B}$, if Θ is close to 1, then $\mathsf{Eval}(\boldsymbol{B}, \Theta)$ considers that decreasing the sum of squared lengths of the orthogonal basis vectors is more important, otherwise decreasing the lexicographical order is more important. Eval balances the two orders and yields a flexible and efficient basis reduction strategy for solving the SVP.

5.2 Parallelization Strategy

Now, we will explain parallelizing the computations method of our lattice reduction algorithm. If the parallel processes have the same lattice basis and generate the same set of lattice vectors, these parallel computations would be useless. As stated in Section 4, we generate different lattice bases for each process. However, if parallel processes have completely different lattice bases, the parallel computation cannot be cooperative.

Fukase and Kashiwabara [20] proposed a parallel computation method such that the basis vectors in first consecutive indices are the same among the processes and the basis vectors in last consecutive indices are different among the processes. Since the basis vectors at last consecutive indices are different, the same set of natural number representations generates a different set of lattice vectors. However, they did not describe an effective parallel computation for more than 100 processes. We propose an effective method for parallel computations, in which each process has same first consecutive basis vectors but different last consecutive basis vectors. In our method, the computational cost of synchronization is practically small. Our method consists of two parts: a part in which each process has different last consecutive basis vectors, and method part in which each process has common first consecutive basis vectors.

Lattice Basis Variants Generation. Here, we explain our new lattice reduction procedure with process-unique information in order to generate many lattice basis variants that have different basis vectors in last consecutive indices. Each process executes this procedure independently with process-unique information `pui`. The procedure is shown in Algorithm 2.

The discontinue rule at line 16 in Algorithm 2 is important. This rule depends on the process-unique information `pui` such that each process executes different lattice basis reduction operations for last consecutive indices of the lattice basis. Thanks to this procedure, a huge number of processes can generate many lattice basis variants even if they have the same lattice basis without communicating with each other.

Cooperation between Many Processes. As mentioned above, one can make a large amount of processes whose lattice bases have different basis vectors in last consecutive indices. Since these processes have the same basis vectors in first consecutive indices from 1, they utilize the stock vectors they have in common. Fukase and Kashiwabara [20] proposed a method to keep this situation by using stock vectors, as explained in Section 3.2; however, it is difficult to maintain for a large number of processes solving higher dimensional problems without imposing a heavy synchronization penalty.

We propose a method to solve the above problem. In it, all the processes share additional global storage, and the basis vectors of the lattice basis possessed by each process are stored in it in the same manner as stock vectors but with a different strategy as explained next. We call the stored lattice vectors *link vectors*. When each process has a new lattice basis, its basis vectors are stored as link vectors. After each process loads link vectors from their storage, it uses them to reduce the lattice basis to the smallest one in the lexicographical order. In this method, the synchronization penalty (computational cost) consists of storing and loading link vectors from storage, but this penalty is practically small.

Note that we can adjust the parameter δ_{stock} in such a way that quite a few stock vectors are found in one process and one main loop evaluation, and we run around thousand processes in parallel, each of which stores stock vectors independently. However, the number of stored stock vectors at any given time is not that large, as we do not keep all stock vectors. This is because old stock vectors are relatively longer than orthogonal basis vectors possessed by running processes, and it is therefore not necessary to store these, and we throw them away. This parameter adjustment is also explained in next subsection. The same strategy was used for the link vectors.

5.3 Parameter Choice

Our algorithm takes a basis $\boldsymbol{B}$ and process-unique information `pui` with parameters ℓ_{fc}, ℓ_{lc}, δ, δ', δ'', δ_{stock}, ℓ_{link}, Θ, and two sets S and S' of NNRs used in line 5 in Algorithm 1 and line 4 in Algorithm 2 as input. $\boldsymbol{B}$ is given by a user, and `pui` is given by a user or a facility, e.g., MPI libraries and a job scheduler, but choosing the remaining parameters is not trivial. We did not find an optimal choice of parameters because, to the best of our knowledge, basic theories allowing this have not been developed. In this section, we explain a method of parameter choice for our algorithm. Note that the concrete chosen values of the parameters when we solved a 150-dimensional problem instance appear in the next section.

More precisely, we observed intermediate values in the main loop of our algorithm (lines 2-14 in Algorithm 1) with input a basis (e.g., 150-dimensional problem instance in the SVP Challenge). The intermediate values which we observed are as follows: (1) The number of found stock vectors. (2) The sum of squared lengths of the orthogonal basis vectors. (3) The calculation times of the two lattice vector generations in the main loop called at lines 4 and 5 in Algorithm 1.

We want that (1) is quite small because we are only interested in relatively short lattice vectors, and adjusting this can be immediately done by decreasing δ_{stock}.

We adjust parameters ℓ_{fc}, ℓ_{lc}, ℓ_{link}, and Θ to appropriate values such that the processes running in parallel can keep a small value of (2) when reducing the basis. As mentioned in Section 5.1, it is quite hard to determine the value of Θ, so that we choose and adjust the parameters by observations regarding the performance and the intermediate values of the main loop of our algorithm.

Regarding (3), we adjust ℓ_{lc}, δ, δ', δ'', the set S of NNRs at line 5 in Algorithm 1, and the set S' of NNRs at line 4 in Algorithm 2. Our algorithm has two subroutines to generate lattice vectors by using S and S' (in line 5 in Algorithm 1 and line 4 in Algorithm 2), and they have different purposes. Since these lattice vector generations

Table 1. Specifications of cluster system Oakleaf-FX to solve the 134-dimension SVP Challenge; note that the units of "CPU frequency" and "Total RAM" are GHz and GB, respectively

CPU	CPU frequency	# of nodes	Total # of cores	Total RAM
SPARC64 IXfx	1.848	6	96	192

are the most costly parts of our algorithm, it is important to optimize the choices of S and S'; however, these sets differ only in their size. Note that NNRs in S and S' are chosen by ascending order of expectation of squared lengths (shown in Theorem 2) of generated lattice vectors with several bases.

The purpose of the lattice vector generation with S (at line 5 in Algorithm 1) is finding relatively short lattice vectors, so the size of S should be large. The purpose of the other lattice vector generation with S' (at line 4 in Algorithm 2) is decreasing the sum of squared length of orthogonal basis vectors and generating many lattice bases variants as mentioned in Section 5.2 by repeatedly applying lattice reduction (e.g., the LLL algorithm), so it is not necessary to choose the size of S' as large.

Actually, we adjust the parameters ℓ_{lc}, δ, δ', δ'', and sizes of sets S and S' in such a way that the calculation times of two lattice vector generations with S and S' are around several minutes. More concretely, we chose the parameters such that the calculation times are 5 minutes and 10 minutes, respectively, and the lattice vector generation with S' is repeatedly evaluated around 500 times per one Algorithm 2 evaluation on average when we found a solution of a 150-dimensional problem instance of the SVP Challenge.

6 Application to SVP Challenge

The shortest vector problem (SVP) is an important problem because of its relation to the security strength of lattice-based cryptosystems, and studying algorithms for solving it is an important research topic. The SVP Challenge was started in 2010 [45] as a way of advancing this research. The challenge provides a problem instance generator and accepts submissions of discovered short lattice vectors. As mentioned in Section 2.3, a submitted lattice vector of a lattice L is inducted into the hall-of-fame if its length is less than $1.05 \cdot \mathrm{GH}(L)$ and less than the lengths of other registered lattice vectors if they exist for the same dimension.

6.1 Equipment

We applied our algorithm to the problems in the SVP Challenge. In this section, we explain our equipment and results.

First, let us describe our equipment. We used four clusters, called Oakleaf-FX (FX10), Chimera, ASGC, and Reedbush-U. The hardware specifications of these clusters are summarized in Tables 1, 2, 3, and 4. The FX10 cluster included a customized compiler, MPI library, and job scheduler. The Chimera, ASGC, and Reedbush-U clusters included compilers, MPI libraries, and job schedulers. In particular, we used GCC version 4.8.4, OpenMPI version 1.8.1, and Univa Grid Engine version 8.1.7 in the Chimera,

Table 2. Specifications of cluster system Chimera to solve SVP Challenge instances of 138, 140, 142, 144, 146, 148, and 150 dimensions; note that the units of "CPU frequency" and "Total RAM" are GHz and GB, respectively

CPU	CPU frequency	# of nodes	Total # of cores	Total RAM
Xeon E5540	2.53	80	640	6240
Xeon X5650	2.66	11	132	765
Xeon E5-2670	2.6	4	64	762
Xeon E5-2695 v3	2.3	2	56	256
Xeon E7-4870	2.4	1	80	4096
Grand Total		98	972	12 119

Table 3. Specifications of cluster system ASGC to solve 148 and 150-dimensions instances of SVP Challenge; note that the units of "CPU frequency" and "Total RAM" are GHz and GB, respectively

CPU	CPU frequency	# of nodes	Total # of cores	Total RAM
Xeon E5-2680 v2	2.8	5	100	640

Intel C++ Compiler version 14.0.2, Intel MPI Library version 4.1, and TORQUE version 4.2.8 in the ASGC, and Intel C++ Compiler version 16.0.3, Intel MPI Library version 5.1, and PBS Professional 13.1 in the Reedbush-U. Note that Hyper-Threading was enabled and Turbo Boost was disabled in all nodes of Chimera, while Hyper-Threading was disabled and Turbo Boost was disabled in all nodes of ASGC and Reedbush-U. Note also that these clusters were shared systems, so all the processes were controlled by the job schedulers. Now, let us briefly explain the pre-processing. Before executing processes for our algorithm, the lattice basis of the input was pre-processed. We used the BKZ algorithm implemented by fplll [50] to reduce components of the lattice basis to small integers.

For each process when we solve the problem instances, the process-unique information `pui` was given as the rank by the MPI libraries and job schedulers, namely, the `pui` value ranged from 0 to $m - 1$, where m is the number of requested processes sent to the MPI libraries and job schedulers.

6.2 Experimental Results

We used our algorithm to solve SVP Challenge problem instances of 134, 138, 140, 142, 144, 146, 148, and 150 dimensions with seed 0. We used FX10 to solve the 134-dimensional instance, Chimera to solve the instances with 138–146 dimensions, and the two clusters Chimera and ASGC to solve the 148-dimensional instance.

To solve the 150-dimensional instance, we used the following numbers of cores and CPUs of the four clusters: (1) 28 cores of Xeon E5-2695 v3 and (2) 80 cores of Xeon E7-4870 in the parts of Chimera, (3) 100 cores of Xeon E5-2680 v2 in ASGC, (4) 192 cores of SPARC64 IXfx per job in FX10, and (5) 864 cores of Xeon E5-2695 v4 per job in Reedbush-U, for (1) 49 days, (2) 81 days, (3) 39 days, (4) 37 days, and (5) 188 days,

Table 4. Specifications of cluster system Reedbush-U to solve the 150-dimensional instance of the SVP Challenge; note that the units of "CPU frequency" and "Total RAM" are GHz and GB, respectively

CPU	CPU frequency	# of nodes	Total # of cores	Total RAM
Xeon E5-2695 v4	2.1	24	864	5856

Table 5. Our experimental results for the SVP Challenge instances of dimensions 134, 138, 140, 142, 144, 146, 148, and 150, where "GH(L)" is the Gaussian heuristic of a given lattice L, "$\boldsymbol{v}$" in the column header indicates a solution of the corresponding row, the "Max. # of requested processes" column shows the maximum number of requested processes sent to the job schedulers, and the "Days to solve" column lists the wall-clock times to obtain each solution

Dimension	$\lfloor \mathrm{GH}(L) \rceil$	$\lfloor \lVert \boldsymbol{v} \rVert \rceil$	$\frac{\lVert \boldsymbol{v} \rVert}{\mathrm{GH}(L)}$	Max. # of requested processes	Days to solve	Root Hermite factor
150	3090	3220	1.041 92	864	394	1.007 68
148	3070	3178	1.035 12	1000	256	1.007 69
146	3056	3195	1.045 34	1000	64	1.007 83
144	3025	3154	1.042 84	700	50	1.007 87
142	3003	3141	1.046 09	700	50	1.007 96
140	2991	3025	1.011 39	500	50	1.007 78
138	2972	3077	1.035 16	500	140	1.008 01
134	2927	2976	1.016 95	92	72	1.008 01

respectively. We did not use these computing nodes simultaneously. Therefore, the total wall-clock time was about 394 days. Note that when we used the computing node (1), we requested 24 processes for job schedulers and the other cases (2), (3), (4), and (5), and the number of requested processes was the same as the number of cores.

When we found a solution of the 150-dimensional instance, tuned parameters of our algorithm were as follows: $\ell_{\mathrm{fc}} = 50$, $\ell_{\mathrm{lc}} = 50$, $\delta = 0.9999$, $\delta' = 0.985$, $\delta'' = 0.0002$, $\delta_{\mathrm{stock}} = 1.08$, $\ell_{\mathrm{link}} = 50$, and $\Theta = 0.93$. The method we used to choose these parameters is explained in Section 5.3.

We clarify the numbers of lattice vectors generated by the two subroutines in line 5 in Algorithm 1 with a set S of NNRs and in line 4 in Algorithm 2 with a set S' of NNRs, and the costs per generation when we solved the 150-dimensional problem and problems with a smaller dimensions less than 150. The sizes of S and S' are around 30 millions NNRs and 50 thousands NNRs, respectively. Note that lines 3-19 in Algorithm 2 are repeated until the termination condition in line 18 is satisfied, and the number of loops is 500 on average. The calculation times of the lattice vector generations with S and S' are around 5 minutes and 10 minutes, respectively. That is, the calculation time of these generations per lattice vector is around 12-20 micro-seconds.

As mentioned in Section 5.2, it is not necessary to store all the stock vectors and link vectors, because old stock vectors and link vectors are relatively longer, so that we threw away old stock vectors and link vectors, and we cannot show the total size. The size of

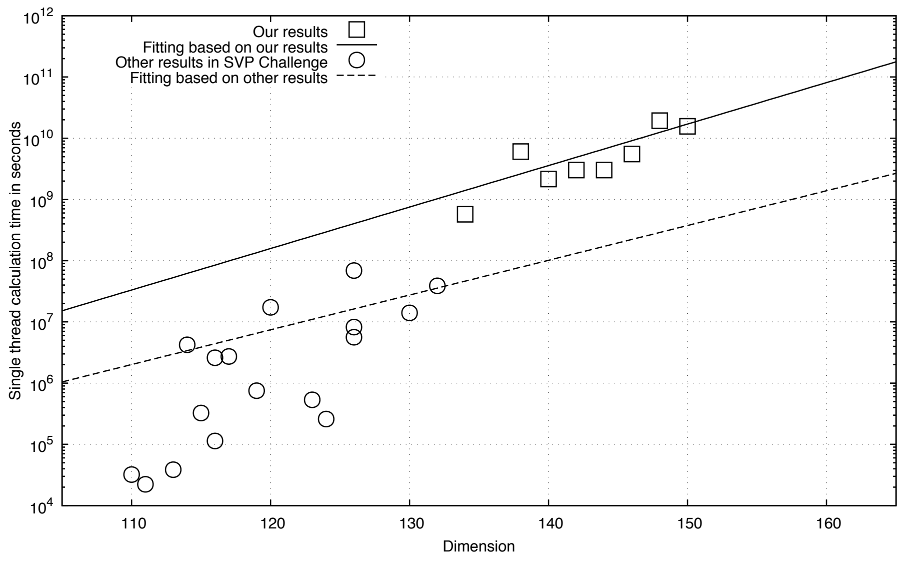

Fig. 1. Comparison of solved dimension and single thread calculation time in seconds of our results and results in the SVP Challenge, where plotted boxes are our 8 results, the straight solid line is fitted to our results, plotted circles are other 17 results of problems with dimensions greater than or equal to 100 and specified overall calculation times, and the dotted line is fitted to these results

the remaining stock vectors and link vectors is around 60 gigabytes. We estimate that the total size of all the found stock vectors and link vectors is several hundred gigabytes for solving the 150-dimensional problem.

Table 5 lists the details of our solutions. The solution of the 150-dimensional problem is the current world record.

We show a comparison of solved dimension and single thread calculation time in seconds of our results and the results from the SVP Challenge in Figure 1. In this figure, plotted boxes are our 8 results, and plotted circles are other 17 results for problems with dimensions greater than or equal to 100 and specified overall calculation times. It is clear that our algorithm outperforms others. This is experimental evidence that our proposal is effective at reducing higher-dimensional lattice bases and solving higher-dimensional SVPs.

Additionally, we show fittings of our and the SVP Challenge results in Figure 1 obtained by `fit` function of GNU Plot version 5.2. As a result, we obtained the straight solid line for our results and the dotted line for the other SVP Challenges results. Note that these lines are expressed as $f(x) = c^{ax+b} + d$ with four variables a, b, c, d. Obtained values and asymptotic standard errors of a, b, c, d, and values of the root mean square (RMS) of residuals by using `fit` are as follows: $a = 1.002\,87 \pm 933.3$, $b = 0.999\,973 \pm 7822$, $c = 1.168\,33 \pm 148.9$, and $d = 1.000\,93 \pm 2.948 \times 10^{10}$, and the value of RMS is $4.852\,06 \times 10^9$ for the straight solid line, and $a = 0.957\,735 \pm 843.9$,

$b = 0.988\,369 \pm 5095$, $c = 1.146\,25 \pm 123.6$, and $d = 1.0004 \pm 5.792 \times 10^7$, and the value of RMS is $1.666\,35 \times 10^7$ for the dotted line.

7 Conclusion

We proposed an algorithm suitable for parallel computing environments to reduce the lattice basis and presented its results in the SVP Challenge.

Regarding the results in the SVP Challenge, we discovered solutions to problem instances of dimensions 134, 138, 140, 142, 144, 146, 148, and 150, a world record for the highest dimension. This is clear experimental evidence that our proposal is effective at reducing the lattice basis and solving the SVP by using a parallel computing environment.

Acknowledgements. A part of this work is supported by JST CREST grant number JPMJCR1688.

References

1. Agrawal, S., Boneh, D., Boyen, X.: Efficient lattice (H)IBE in the standard model. In: Gilbert [25], pp. 553–572, `http://dx.doi.org/10.1007/978-3-642-13190-5_28`
2. Agrawal, S., Freeman, D.M., Vaikuntanathan, V.: Functional encryption for inner product predicates from learning with errors. In: Lee and Wang [35], pp. 21–40, `http://dx.doi.org/10.1007/978-3-642-25385-0_2`
3. Ajtai, M.: The shortest vector problem in L_2 is *NP*-hard for randomized reductions (extended abstract). In: Vitter, J.S. (ed.) Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998. pp. 10–19. ACM (1998), `http://doi.acm.org/10.1145/276698.276705`
4. Ajtai, M., Dwork, C.: A public-key cryptosystem with worst-case/average-case equivalence. In: Leighton, F.T., Shor, P.W. (eds.) Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997. pp. 284–293. ACM (1997), `http://doi.acm.org/10.1145/258533.258604`
5. Aono, Y., Nguyen, P.Q.: Random sampling revisited: Lattice enumeration with discrete pruning. In: Coron, J., Nielsen, J.B. (eds.) Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10211, pp. 65–102 (2017), `http://dx.doi.org/10.1007/978-3-319-56614-6_3`
6. Aono, Y., Wang, Y., Hayashi, T., Takagi, T.: Improved progressive BKZ algorithms and their precise cost estimation by sharp simulator. In: Fischlin and Coron [19], pp. 789–819, `http://dx.doi.org/10.1007/978-3-662-49890-3_30`
7. Boneh, D., Gentry, C., Gorbunov, S., Halevi, S., Nikolaenko, V., Segev, G., Vaikuntanathan, V., Vinayagamurthy, D.: Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In: Nguyen, P.Q., Oswald, E. (eds.) Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8441, pp. 533–556. Springer (2014), `http://dx.doi.org/10.1007/978-3-642-55220-5_30`

8. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. In: Goldwasser, S. (ed.) Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012. pp. 309–325. ACM (2012), http://doi.acm.org/10.1145/2090236.2090262
9. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. TOCT 6(3), 13:1–13:36 (2014), http://doi.acm.org/10.1145/2633600
10. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. In: Ostrovsky, R. (ed.) IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011. pp. 97–106. IEEE Computer Society (2011), http://dx.doi.org/10.1109/FOCS.2011.12
11. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. SIAM J. Comput. 43(2), 831–871 (2014), http://dx.doi.org/10.1137/120868669
12. Brakerski, Z., Vaikuntanathan, V.: Lattice-based FHE as secure as PKE. In: Naor, M. (ed.) Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12-14, 2014. pp. 1–12. ACM (2014), http://doi.acm.org/10.1145/2554797.2554799
13. Buchmann, J., Ludwig, C.: Practical lattice basis sampling reduction. Cryptology ePrint Archive, Report 2005/072 (2005), https://eprint.iacr.org/2005/072, https://eprint.iacr.org/2005/072
14. Buchmann, J.A., Ludwig, C.: Practical lattice basis sampling reduction. In: Hess, F., Pauli, S., Pohst, M.E. (eds.) Algorithmic Number Theory, 7th International Symposium, ANTS-VII, Berlin, Germany, July 23-28, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4076, pp. 222–237. Springer (2006), http://dx.doi.org/10.1007/11792086_17
15. Chen, Y., Nguyen, P.Q.: BKZ 2.0: Better lattice security estimates. In: Lee and Wang [35], pp. 1–20, http://dx.doi.org/10.1007/978-3-642-25385-0_1
16. Dagdelen, Ö., Schneider, M.: Parallel enumeration of shortest lattice vectors. In: D'Ambra, P., Guarracino, M.R., Talia, D. (eds.) Euro-Par 2010 - Parallel Processing, 16th International Euro-Par Conference, Ischia, Italy, August 31 - September 3, 2010, Proceedings, Part II. Lecture Notes in Computer Science, vol. 6272, pp. 211–222. Springer (2010), http://dx.doi.org/10.1007/978-3-642-15291-7_21
17. Detrey, J., Hanrot, G., Pujol, X., Stehlé, D.: Accelerating lattice reduction with fpgas. In: Abdalla, M., Barreto, P.S.L.M. (eds.) Progress in Cryptology - LATINCRYPT 2010, First International Conference on Cryptology and Information Security in Latin America, Puebla, Mexico, August 8-11, 2010, Proceedings. Lecture Notes in Computer Science, vol. 6212, pp. 124–143. Springer (2010), http://dx.doi.org/10.1007/978-3-642-14712-8_8
18. Dwork, C. (ed.): Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008. ACM (2008)
19. Fischlin, M., Coron, J. (eds.): Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I, Lecture Notes in Computer Science, vol. 9665. Springer (2016), http://dx.doi.org/10.1007/978-3-662-49890-3
20. Fukase, M., Kashiwabara, K.: An accelerated algorithm for solving SVP based on statistical analysis. JIP 23(1), 67–80 (2015), http://dx.doi.org/10.2197/ipsjjip.23.67
21. Gama, N., Nguyen, P.Q.: Finding short lattice vectors within mordell's inequality. In: Dwork [18], pp. 207–216, http://doi.acm.org/10.1145/1374376.1374408
22. Gama, N., Nguyen, P.Q.: Predicting lattice reduction. In: Smart, N.P. (ed.) Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings. Lecture Notes in Computer Science, vol. 4965, pp. 31–51. Springer (2008), http://dx.doi.org/10.1007/978-3-540-78967-3_3

23. Gama, N., Nguyen, P.Q., Regev, O.: Lattice enumeration using extreme pruning. In: Gilbert [25], pp. 257–278, http://dx.doi.org/10.1007/978-3-642-13190-5_13
24. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Dwork [18], pp. 197–206, http://doi.acm.org/10.1145/1374376.1374407
25. Gilbert, H. (ed.): Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings, Lecture Notes in Computer Science, vol. 6110. Springer (2010), http://dx.doi.org/10.1007/978-3-642-13190-5
26. Goldreich, O., Goldwasser, S., Halevi, S.: Public-key cryptosystems from lattice reduction problems. In: Jr., B.S.K. (ed.) Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings. Lecture Notes in Computer Science, vol. 1294, pp. 112–131. Springer (1997), http://dx.doi.org/10.1007/BFb0052231
27. Gorbunov, S., Vaikuntanathan, V., Wee, H.: Attribute-based encryption for circuits. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013. pp. 545–554. ACM (2013), http://doi.acm.org/10.1145/2488608.2488677
28. Gorbunov, S., Vaikuntanathan, V., Wee, H.: Attribute-based encryption for circuits. J. ACM 62(6), 45 (2015), http://doi.acm.org/10.1145/2824233
29. Gorbunov, S., Vaikuntanathan, V., Wee, H.: Predicate encryption for circuits from LWE. In: Gennaro, R., Robshaw, M. (eds.) Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9216, pp. 503–523. Springer (2015), http://dx.doi.org/10.1007/978-3-662-48000-7_25
30. Hermans, J., Schneider, M., Buchmann, J.A., Vercauteren, F., Preneel, B.: Parallel shortest lattice vector enumeration on graphics cards. In: Bernstein, D.J., Lange, T. (eds.) Progress in Cryptology - AFRICACRYPT 2010, Third International Conference on Cryptology in Africa, Stellenbosch, South Africa, May 3-6, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6055, pp. 52–68. Springer (2010), http://dx.doi.org/10.1007/978-3-642-12678-9_4
31. Hoffstein, J., Pipher, J., Silverman, J.H.: NTRU: A ring-based public key cryptosystem. In: Buhler, J. (ed.) Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings. Lecture Notes in Computer Science, vol. 1423, pp. 267–288. Springer (1998), http://dx.doi.org/10.1007/BFb0054868
32. Hoffstein, J., Pipher, J., Silverman, J.H.: An Introduction to Mathematical Cryptography. Undergraduate Texts in Mathematics, Springer-Verlag, 1st edn. (2008), http://dx.doi.org/10.1007/978-0-387-77993-5
33. Kannan, R.: Improved algorithms for integer programming and related lattice problems. In: Johnson, D.S., Fagin, R., Fredman, M.L., Harel, D., Karp, R.M., Lynch, N.A., Papadimitriou, C.H., Rivest, R.L., Ruzzo, W.L., Seiferas, J.I. (eds.) Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA. pp. 193–206. ACM (1983), http://doi.acm.org/10.1145/800061.808749
34. Kuo, P., Schneider, M., Dagdelen, Ö., Reichelt, J., Buchmann, J.A., Cheng, C., Yang, B.: Extreme enumeration on GPU and in clouds - - how many dollars you need to break SVP challenges -. In: Preneel and Takagi [43], pp. 176–191, http://dx.doi.org/10.1007/978-3-642-23951-9_12
35. Lee, D.H., Wang, X. (eds.): Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings, Lecture Notes in Computer Science, vol. 7073. Springer (2011), http://dx.doi.org/10.1007/978-3-642-25385-0

36. Lenstra, A.K., Lenstra, H.W., Lovász, L.: Factoring polynomials with rational coefficients. Mathematische Annalen 261(4), 515–534 (1982), http://dx.doi.org/10.1007/BF01457454
37. Ludwig, C.: Practical Lattice Basis Sampling Reduction. Ph.D. thesis, Technische Universität Darmstadt Universitäts (2005), http://elib.tu-darmstadt.de/diss/000640
38. Micciancio, D., Walter, M.: Fast lattice point enumeration with minimal overhead. In: Indyk, P. (ed.) Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015. pp. 276–294. SIAM (2015), http://dx.doi.org/10.1137/1.9781611973730.21
39. Micciancio, D., Walter, M.: Practical, predictable lattice basis reduction. In: Fischlin and Coron [19], pp. 820–849, http://dx.doi.org/10.1007/978-3-662-49890-3_31
40. Nguyen, P.Q., Vallée, B. (eds.): The LLL Algorithm - Survey and Applications. Information Security and Cryptography, Springer (2010), http://dx.doi.org/10.1007/978-3-642-02295-1
41. Pieterse, V., Black, P.E.: “single program multiple data” in Dictionary of Algorithms and Data Structures (December 2004), http://www.nist.gov/dads/HTML/singleprogrm.html
42. Plantard, T., Schneider, M.: Creating a challenge for ideal lattices. Cryptology ePrint Archive, Report 2013/039 (2013), https://eprint.iacr.org/2013/039
43. Preneel, B., Takagi, T. (eds.): Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings, Lecture Notes in Computer Science, vol. 6917. Springer (2011)
44. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005. pp. 84–93. ACM (2005), http://doi.acm.org/10.1145/1060590.1060603
45. Schneider, M., Gama, N.: SVP Challenge, http://www.latticechallenge.org/svp-challenge/
46. Schneider, M., Göttert, N.: Random sampling for short lattice vectors on graphics cards. In: Preneel and Takagi [43], pp. 160–175, http://dx.doi.org/10.1007/978-3-642-23951-9_11
47. Schnorr, C.: Lattice reduction by random sampling and birthday methods. In: Alt, H., Habib, M. (eds.) STACS 2003, 20th Annual Symposium on Theoretical Aspects of Computer Science, Berlin, Germany, February 27 - March 1, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2607, pp. 145–156. Springer (2003), http://dx.doi.org/10.1007/3-540-36494-3_14
48. Schnorr, C.: Correction to lattice reduction by random sampling and birthday methods. http://www.math.uni-frankfurt.de/~dmst/research/papers.html (February 2012)
49. Schnorr, C., Euchner, M.: Lattice basis reduction: Improved practical algorithms and solving subset sum problems. Math. Program. 66, 181–199 (1994), http://dx.doi.org/10.1007/BF01581144
50. The FPLLL development team: fplll, a lattice reduction library (2016), available at https://github.com/fplll/fplll