

NARIAKI TATEIWA, YUJI SHINANO, MASAYA YASUDA, SHIZUO KAJI,  
KEIICHIRO YAMAMURA, KATSUKI FUJISAWA

## **Massively parallel sharing lattice basis reduction**

---

\*The work for this article has been partially conducted within the Research Campus MODAL funded by the German Federal Ministry of Education and Research (fund number 05M20ZBM) and the project HPO-Navi (fund number 391087700): Sustainable Infrastructures for Archiving and Publishing High-Performance Optimization Software

Zuse Institute Berlin  
Takustr. 7  
14195 Berlin  
Germany

Telephone: +49 30 84185-0  
Telefax: +49 30 84185-125

E-mail: [bibliothek@zib.de](mailto:bibliothek@zib.de)  
URL: <http://www.zib.de>

ZIB-Report (Print) ISSN 1438-0064  
ZIB-Report (Internet) ISSN 2192-7782

# Massively parallel sharing lattice basis reduction

Nariaki Tateiwa,<sup>1</sup> Yuji Shinano,<sup>2</sup> Masaya Yasuda,<sup>3</sup> Shizuo Kaji,<sup>4</sup>  
Keiichiro Yamamura,<sup>5</sup> Katsuki Fujisawa<sup>6</sup>

December 9, 2021

## Abstract

For cryptanalysis in lattice-based schemes, the performance evaluation of lattice basis reduction using high-performance computers is becoming increasingly important for the determination of the security level. We propose a distributed and asynchronous parallel reduction algorithm based on randomization and DeepBKZ, which is an improved variant of the block Korkine-Zolotarev (BKZ) reduction algorithm. Randomized copies of a lattice basis are distributed to up to 103,680 cores and independently reduced in parallel, while some basis vectors are shared asynchronously among all processes via MPI. There is a trade-off between randomization and information sharing; if a substantial amount of information is shared, all processes will work on the same problem, thereby diminishing the benefit of parallelization. To monitor this balance between randomness and sharing, we propose a metric to quantify the variety of lattice bases. We empirically find an optimal parameter of sharing for high-dimensional lattices. We demonstrate the efficacy of our proposed parallel algorithm and implementation with respect to both performance and scalability through our experiments.

## 1 Introduction

A *lattice* is a discrete subgroup of the real vector space  $\mathbb{R}^n$  and is defined as the set of all integral linear combinations of some linearly independent vectors, which is called a *basis* of the lattice. The number of basis vectors is called the *dimension* of the lattice. *Lattice problems* are algorithmic problems related to lattices, and they include a number of hard

---

<sup>1</sup>Graduate School of Mathematics, Kyushu University, Fukuoka, 819-0395, Japan

<sup>2</sup>Department of Applied Algorithmic Intelligence Methods (A<sup>2</sup>IM), Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany

<sup>3</sup>Department of Mathematics, Rikkyo University, Tokyo, 171-8501, Japan

<sup>4</sup>Institute of Mathematics for Industry, Kyushu University, 819-0395, Japan

<sup>5</sup>Graduate School of Mathematics, Kyushu University, Fukuoka, 819-0395, Japan

<sup>6</sup>Institute of Mathematics for Industry, Kyushu University, 819-0395, Japan

problems such as the *shortest vector problem (SVP)* that asks us to find a non-zero shortest lattice vector given a basis. Over the past decade, lattices have been actively used to construct various cryptosystems, such as fully homomorphic encryption and post-quantum cryptography (PQC). In particular, the security of most lattice-based PQC candidates is fundamentally based on the hardness of solving the SVP (see [4]).

Many studies have been reported on algorithms for solving lattice problems. *Lattice basis reduction* is a powerful tool to solve lattice problems, including SVP. Reduction algorithms may not necessarily find shortest lattice vectors, but they are significantly faster than exact-SVP algorithms such as enumeration and sieve (see [25, 38] for a survey). Given a lattice basis, the aim of lattice basis reduction is to find a new basis of the same lattice consisting of relatively shorter vectors that are nearly-orthogonal. Most lattice problems are easier to solve with such a reduced basis. The Lenstra-Lenstra-Lovász (LLL) algorithm [24] is the most celebrated algorithm, and its blockwise generalization is the block Korkine-Zolotarev (BKZ) algorithm [31]. Recently, efficient variants of BKZ such as BKZ 2.0 [11] have been implemented in software libraries (e.g., `fpLLL` library [36]), and they have been used to estimate the security level of lattice-based cryptosystems (e.g., see [2, 4]). In contrast, it is mandatory to consider the effect of large-scale parallelization when solving algorithms in cryptanalysis (see [21]). In this study, we consider the large-scale parallelization of reduction algorithms. In particular, we use DeepBKZ [37], an enhancement of BKZ, for our software development.

## 1.1 Previous work on the large-scale parallelization of reduction algorithms

While there are many reports of parallelization for enumeration and sieve algorithms (Subsection 1.3 below), to date, few studies for reduction algorithms have been conducted. In 2020, a distributed and asynchronous parallel reduction algorithm was first developed in [33], which is called **MAP-SVP** (MAssively Parallel solver for SVP). It was built on the Ubiquity Generator (UG) framework [32], a generic framework for branch-and-bound algorithms, to parallelize a reduction algorithm based on randomization that generates different bases of the same lattice through unimodular transformation of an input basis. Specifically, **MAP-SVP** runs a reduction algorithm (e.g., BKZ or DeepBKZ) on each solver independently for a randomized basis while also enabling the sharing of the shortest basis vector with all solvers to accelerate the reduction process for each of them. The performance and scalability of **MAP-SVP** were reported in [33, Section V] by using up to 100,032 cores for solving several instances of the Darmstadt SVP challenge [28]. In 2021, a generic framework of parallelization was proposed in [34] for lattice algorithms, and it is called **CMAP-LAP** (Configurable Massively Parallel Solver for Lattice Problems). This framework covers the parallelization of reduction, enumeration, and sieve algorithms and can run these algorithms cooperatively on a large-scale computational platform. **CMAP-LAP** was built on a generalized UG (UG version 1.0 RC) from scratch, and it performs a parallel execution of the *supervisor-worker* style [27]. Given an input instance, a supervisor

process distributes randomized instances as tasks to all workers, and each worker process can execute multiple kinds of solvers and multi-threads solvers in a heterogeneous manner. In addition, the supervisor stores lattice bases and vectors in data containers. Using these data containers, each solver can share a lattice basis and vectors asynchronously with minimal communication overhead. The stability of CMAP-LAP was demonstrated by conducting large-scale experiments.

## 1.2 Our contribution

In this paper, we develop software specialized for the massive parallelization of lattice basis reduction. Specifically, we parallelize DeepBKZ [37] in the CMAP-LAP framework and call our software CMAP-DeepBKZ. (Note that BKZ can also be adopted in the same way.) Below we summarize our contribution:

- While MAP-SVP [33] only shares a single short lattice vector among solvers, CMAP-DeepBKZ can share *multiple* short vectors to accelerate the reduction process for every solver more efficiently than MAP-SVP. In CMAP-DeepBKZ, each solver periodically sends its short basis vectors to a data container in the supervisor. In contrast, the supervisor only distributes short lattice vectors from its container to the solver if they affect the solver’s reduction algorithm. Thus, every solver can share short lattice vectors with the other solvers while only directly communicating with the supervisor.
- While the reduction process can accelerate for each solver as more vectors are shared, the randomness of the bases processed by the solvers might be lost. Therefore we propose a method to quantify the diversity of lattice bases using metrics for Grassmann manifolds (e.g., see [5, 6] for Grassmann metrics). Using this method, we inspect the randomness of the bases of our parallel reduction algorithm in CMAP-DeepBKZ through experimentation.
- We demonstrate the performance and the scalability of CMAP-DeepBKZ by conducting large-scale experiments using up to 103,680 cores. Specifically, we evaluate how the discovery of short lattice vectors and the quality of the output basis change depending on the numbers of shared vectors and CPU cores. We also evaluate the application performance of CMAP-DeepBKZ such as the CPU utilization in a large-scale computing environment. For our experiments, we use instances of the Darmstadt SVP challenge [28] for up to 132 dimensions.

## 1.3 Other work on the parallelization of lattice algorithms

We summarize studies other than those listed in Subsection 1.1 that relate to the parallelization of lattice algorithms. The simplest strategy is the divide-and-conquer method for exact-SVP algorithms such as enumeration and sieve, accomplished by dividing their search space (e.g., see [12, 20, 23]). Randomization is another way for parallelization, and it is useful for pruned enumeration of [18]. Because pruned enumeration trees change

for different bases, a parallel enumeration search can be conducted by performing pruned enumeration on different bases (for more detail, see [8] for a shared-memory parallel enumeration system using randomization and pruning techniques). In 2018, Teruya et al. [35] proposed a massive parallelization method for random sampling. In their system, basis vectors except the last few vectors are stored in global storage and are shared with all processes. Each process performs random sampling independently on its basis and competes to reduce the basis vectors in global storage. Synchronization is only required for storing and loading the basis vectors between each process and global storage. In 2019, Albrecht et al. [3] provided the General Sieve Kernel, abbreviated as **G6K**, supporting a variety of lattice basis reductions using advanced sieve algorithms. For BKZ with **G6K**, we can use a sieve algorithm to run as a core exact-SVP oracle for local block lattices. **G6K** adopts a multi-thread parallelization with a highly optimized implementation for core sieve algorithms in high-dimensional lattices. In 2021, a GPU implementation was provided in [15] for advanced sieve algorithms in **G6K** to break high-dimensional instances in the Darmstadt SVP challenge (see also [26] for a GPU implementation of enumeration). Both works of [3, 35] essentially parallelize core algorithms such as random sampling and sieve to reduce an input basis while avoiding synchronization overhead.

## 2 Mathematical and algorithmic preliminaries on lattices

We present mathematical properties on lattices. We also introduce algorithmic problems for lattices and practical algorithms of lattice basis reduction for solving lattice problems (see, e.g., [7, 25, 38] for details).

### 2.1 Basics on lattices

**Lattices and their bases** A discrete additive subgroup of the  $n$ -dimensional real vector space  $\mathbb{R}^n$  is called a *lattice*. Any lattice in  $\mathbb{R}^n$  is the set of all *integral* combinations of linearly independent vectors  $\mathbf{b}_1, \dots, \mathbf{b}_d$  in  $\mathbb{R}^n$  ( $n \geq d$ ) as

$$\mathcal{L}(\mathbf{b}_1, \dots, \mathbf{b}_d) = \left\{ \sum_{i=1}^d x_i \mathbf{b}_i \in \mathbb{R}^n : x_1, \dots, x_d \in \mathbb{Z} \right\}.$$

The set  $(\mathbf{b}_1, \dots, \mathbf{b}_d)$  of linearly independent vectors spanning a lattice  $L$  is called a *basis* of  $L$ , that is,  $L = \mathcal{L}(\mathbf{b}_1, \dots, \mathbf{b}_d)$ . For convenience, we handle any basis as the  $d \times n$  matrix  $\mathbf{B}$  whose rows are  $\mathbf{b}_i$ 's, and simply write  $L = \mathcal{L}(\mathbf{B})$ . The *dimension* (or *rank*) of  $L$  is defined as the number of basis vectors of  $L$ , denoted by  $\dim(L)$ . A lattice  $L$  in  $\mathbb{R}^n$  is said of *full-rank* when  $n = \dim(L)$ . Two bases  $\mathbf{B}$  and  $\mathbf{C}$  span the same lattice if and only if there exists a unimodular matrix  $\mathbf{T}$  such that  $\mathbf{C} = \mathbf{T}\mathbf{B}$ . This implies that there are infinitely many bases of a lattice  $L$  if  $\dim(L) \geq 2$ . It also shows that for two bases  $\mathbf{B}$  and  $\mathbf{C}$  of a lattice  $L$ , their Gram determinants are both equal. Then the *volume* of  $L$  is defined as

$\text{vol}(L) = \sqrt{\det(\mathbf{B}\mathbf{B}^\top)}$  for any basis  $\mathbf{B}$  of  $L$ . In particular, we have  $\text{vol}(L) = |\det(\mathbf{B})|$  for a full-rank lattice  $L$ .

**Gram-Schmidt orthogonalization** Let  $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_d)$  be a basis of a lattice  $L$ . The *Gram-Schmidt orthogonalization* of  $\mathbf{B}$  is the orthogonal family  $\mathbf{B}^* = (\mathbf{b}_1^*, \dots, \mathbf{b}_d^*)$ , defined recursively by  $\mathbf{b}_1^* = \mathbf{b}_1$  and for  $2 \leq i \leq d$

$$\mathbf{b}_i^* = \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{ij} \mathbf{b}_j^*, \quad \mu_{ij} = \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\|\mathbf{b}_j^*\|^2} \quad (i > j).$$

Let  $\mu = (\mu_{ij})$  denote the  $d \times d$  lower triangular matrix defined by the Gram-Schmidt coefficients with diagonal entries all equal to 1. Then  $\mathbf{B} = \mu \mathbf{B}^*$ , where  $\mathbf{B}^*$  is the  $d \times n$  matrix whose rows are  $\mathbf{b}_i^*$ 's. This shows  $\text{vol}(L) = \prod_{i=1}^d \|\mathbf{b}_i^*\|$  by the orthogonality of Gram-Schmidt vectors.

**Projected lattices** Let  $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_d)$  be a basis of a lattice  $L$ , and  $\mathbf{B}^* = (\mathbf{b}_1^*, \dots, \mathbf{b}_d^*)$  its Gram-Schmidt vectors. For every  $1 \leq k \leq d$ , we define a projection map to the  $\mathbb{R}$ -vector space  $\langle \mathbf{b}_k^*, \dots, \mathbf{b}_d^* \rangle_{\mathbb{R}}$  spanned by  $\mathbf{b}_k^*, \dots, \mathbf{b}_d^*$  as

$$\pi_k : \mathbb{R}^n \longrightarrow \langle \mathbf{b}_k^*, \dots, \mathbf{b}_d^* \rangle_{\mathbb{R}}, \quad \pi_k(\mathbf{v}) = \sum_{i=k}^d \frac{\langle \mathbf{v}, \mathbf{b}_i^* \rangle}{\|\mathbf{b}_i^*\|^2} \mathbf{b}_i^* \quad (\mathbf{v} \in \mathbb{R}^n).$$

The lattice in  $\mathbb{R}^n$  spanned by  $\pi_k(\mathbf{b}_k), \dots, \pi_k(\mathbf{b}_d)$  is called a *projected lattice* of  $L$ , denoted by  $\pi_k(L)$ . The projected lattice  $\pi_k(L)$  has dimension  $d - k + 1$  and volume equal to  $\prod_{i=k}^d \|\mathbf{b}_i^*\|$  since the Gram-Schmidt orthogonalization of  $(\pi_k(\mathbf{b}_k), \dots, \pi_k(\mathbf{b}_d))$  is given by  $\mathbf{b}_k^*, \dots, \mathbf{b}_d^*$ . Note that any projected lattice depends on a basis  $\mathbf{B}$  of  $L$ .

**Successive minima and the Gaussian Heuristic** For  $1 \leq k \leq d$ , the  $k$ -th *successive minimum* of a  $d$ -dimensional lattice  $L$ , denoted by  $\lambda_k(L)$ , is the smallest radius of a ball centered at the origin  $\mathbf{0}$  containing  $k$  linearly independent vectors in  $L$ . In particular, the first minimum  $\lambda_1(L)$  is equal to the length of a non-zero shortest vector in  $L$ . Given a lattice  $L$  of dimension  $d$  and a measurable set  $S$  in  $\mathbb{R}^d$ , the *Gaussian Heuristic* predicts that the number of vectors in  $L \cap S$  is roughly equal to  $\text{vol}(S)/\text{vol}(L)$ . By applying to the ball centered at the origin in  $\mathbb{R}^d$  with radius  $\lambda_1(L)$ , it leads to the prediction of the norm of a shortest non-zero vector in  $L$ . Specifically, the expectation of  $\lambda_1(L)$  according to the Gaussian Heuristic is given by

$$\lambda_1(L) \approx \omega_d^{-\frac{1}{d}} \text{vol}(L)^{\frac{1}{d}} \sim \sqrt{\frac{d}{2\pi e}} \text{vol}(L)^{\frac{1}{d}} =: \text{GH}(L), \quad (1)$$

where  $\omega_d$  denotes the volume of the unit ball in  $\mathbb{R}^d$ . This is only a heuristic, but it roughly holds for “random” lattices of high dimensions  $d \geq 50$ .

## 2.2 Lattice problems: Algorithmic problems for lattices

Of various lattice problems, the *shortest vector problem (SVP)* is the most famous; “Given a basis  $\mathbf{B}$ , find a shortest non-zero vector in the lattice  $L = \mathcal{L}(\mathbf{B})$ , that is, a vector  $\mathbf{s} \in L$  such that  $\|\mathbf{s}\| = \lambda_1(L)$ .” SVP is proven NP-hard under randomized reductions [1]. Approximate factors relax SVP; “Given a basis  $\mathbf{B}$  and an approximation factor  $f \geq 1$ , find a non-zero vector  $\mathbf{v}$  in the lattice  $L = \mathcal{L}(\mathbf{B})$  satisfying  $\|\mathbf{v}\| \leq f\lambda_1(L)$ .” The *closest vector problem (CVP)* is another famous problem; “Given a basis  $\mathbf{B}$  and a target vector  $\mathbf{t}$ , find a vector in  $L = \mathcal{L}(\mathbf{B})$  closest to  $\mathbf{t}$ , that is, a vector  $\mathbf{v} \in L$  such that the distance  $\|\mathbf{t} - \mathbf{v}\|$  is minimized.” It is known that SVP is not harder than CVP. As in the case of SVP, approximate factors relax CVP. Since Kannan’s embedding [22] transforms approximate-CVP into approximate-SVP, both problems seem equally hard in practice.

The security of modern lattice-based cryptosystems is based on the hardness of cryptographic lattice problems such as the LWE and the NTRU problems. Such problems can be reduced to approximate-SVP or/and -CVP (e.g., see [4] for details).

## 2.3 Lattice basis reduction algorithms

We introduce practical reduction algorithms that give a powerful instrument solving lattice problems.

**LLL** We say a basis  $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_d)$  to be  $\delta$ -LLL-reduced for a reduction parameter  $\frac{1}{4} < \delta < 1$  if (i) (size-reduced) it holds  $|\mu_{ij}| \leq \frac{1}{2}$  for all  $i > j$ , and (ii) (Lovász’ condition)  $\delta \|\mathbf{b}_{k-1}^*\|^2 \leq \|\pi_{k-1}(\mathbf{b}_k)\|^2$  for all  $2 \leq k \leq d$ , where  $\mu_{ij}$ ’s and  $\mathbf{b}_k^*$ ’s are Gram-Schmidt coefficients and vectors of  $\mathbf{B}$ , respectively (recall that  $\pi_{k-1}$  denotes the projection map to the  $\mathbb{R}$ -vector space spanned by  $\mathbf{b}_{k-1}^*, \dots, \mathbf{b}_d^*$ ). For a  $\delta$ -LLL-reduced basis  $\mathbf{B}$ , it holds both  $\|\mathbf{b}_1\| \leq \alpha^{\frac{d-1}{2}} \lambda_1(L)$  and  $\|\mathbf{b}_1\| \leq \alpha^{\frac{d-1}{4}} \text{vol}(L)^{\frac{1}{d}}$  for  $L = \mathcal{L}(\mathbf{B})$  and  $\alpha = \frac{4}{4\delta-1}$  (see [7, 25]). To find an LLL-reduced basis, the LLL algorithm [24] calls size-reduction as a subroutine, and it also swaps adjacent basis vectors that do not satisfy Lovász’ condition. The LLL algorithm has complexity polynomial in  $d$ , and it is also useful to get rid of the linear dependency of vectors.

**LLL with deep insertions (DeepLLL)** As a generalization of LLL, *non-adjacent* basis vectors can be changed in DeepLLL [31]; Given a basis  $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_d)$  and a reduction parameter  $\frac{1}{4} < \delta < 1$ , we insert the  $k$ -th basis vector  $\mathbf{b}_k$  before  $\mathbf{b}_i$  as  $\mathbf{B} \leftarrow (\mathbf{b}_1, \dots, \mathbf{b}_{i-1}, \mathbf{b}_k, \mathbf{b}_i, \dots, \mathbf{b}_{k-1}, \mathbf{b}_{k+1}, \dots, \mathbf{b}_d)$  for indexes  $i < k$  such that  $\|\pi_i(\mathbf{b}_k)\|^2 < \delta \|\mathbf{b}_i^*\|^2$ . This basis permutation is called a *deep insertion*. After a deep insertion, the  $i$ -th new Gram-Schmidt vector is given by  $\pi_i(\mathbf{b}_k)$ , whose length is shorter than the old one. We say a basis  $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_d)$  to be  $\delta$ -DeepLLL-reduced if it is size-reduced and  $\delta \|\mathbf{b}_i^*\|^2 \leq \|\pi_i(\mathbf{b}_k)\|^2$  for all  $i < k$ . For a  $\delta$ -DeepLLL-reduced basis  $\mathbf{B}$ , it holds both  $\|\mathbf{b}_1\| \leq \sqrt{\alpha} \left(1 + \frac{\alpha}{4}\right)^{\frac{d-2}{2}} \lambda_1(L)$  and  $\|\mathbf{b}_1\| \leq \alpha^{\frac{d-1}{2d}} \left(1 + \frac{\alpha}{4}\right)^{\frac{(d-1)(d-2)}{4d}} \text{vol}(L)^{\frac{1}{d}}$  for  $L = \mathcal{L}(\mathbf{B})$  and

$\alpha = \frac{4}{4\alpha-1}$  (see [40] for a proof). These properties are better than LLL, but the complexity is no longer polynomial in  $d$ .

**BKZ** For a basis  $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_d)$  of a lattice  $L$ , set

$$\mathbf{B}_{[j,k]} = (\pi_j(\mathbf{b}_j), \pi_j(\mathbf{b}_{j+1}), \dots, \pi_j(\mathbf{b}_k))$$

and  $L_{[j,k]} = \mathcal{L}(\mathbf{B}_{[j,k]})$  for  $j < k$ . For a blocksize  $\beta \geq 2$ , a basis  $\mathbf{B}$  is said to be  $\beta$ -BKZ-reduced if it is size-reduced and  $\|\mathbf{b}_j^*\| = \lambda_1(L_{[j,k]})$  for every  $1 \leq j \leq d-1$  and  $k = \min(j+\beta-1, d)$ . In particular, it is called HKZ-reduced when  $\beta = d$  (e.g., see [2, Definition 3] for the definition of HKZ-reduction). For a  $\beta$ -BKZ-reduced basis  $\mathbf{B}$ , it holds  $\|\mathbf{b}_1\| \leq \gamma_{\beta}^{\frac{d-1}{\beta-1}} \lambda_1(L)$  [29], where  $\gamma_{\beta}$  denotes Hermite's constant of dimension  $\beta$  (see [25] for Hermite's constants). A  $\beta$ -BKZ-reduced basis can be found by the BKZ algorithm [31], in which LLL is called to reduce  $\mathbf{B}_{[j,k]}$  before calling an exact-SVP algorithm (e.g., an enumeration algorithm) over  $L_{[j,k]}$ . Since larger  $\beta$  decreases  $\gamma_{\beta}^{1/(\beta-1)}$  from Mordell's inequality, BKZ finds short lattice vectors, but its computational cost is much more expensive. The complexity of BKZ depends on that of an exact-SVP algorithm over  $L_{[j,k]}$ .

**DeepBKZ** It is an enhancement of BKZ proposed in [37] that uses DeepLLL as a subroutine in a BKZ framework (instead of LLL). We show a basic procedure of DeepBKZ in Algorithm 1 that calls enumeration as an exact-SVP algorithm in Step 7. In practice, DeepBKZ can find shorter lattice vectors than BKZ in using the same blocksize  $\beta$  (see [37, 39] for their experimental results). Similarly to BKZ, the complexity of DeepBKZ depends on that of exact-SVP algorithm (e.g., enumeration) in dimension  $\beta$ .

### 3 Parallelization of lattice basis reduction

In this section, we introduce the massive parallelization system of DeepBKZ and its implementation. Our parallelization is based on randomization, which enables task-parallel reductions for multiple randomized bases. We also share short basis vectors of the randomized bases among solvers to accelerate the reduction process for every solver.

#### 3.1 Ordering of lattice bases for reduction

We define an ordering of lattice bases for reduction. Let us recall the process of DeepBKZ: given a basis of a lattice  $L$  and a blocksize  $\beta \geq 2$ , DeepBKZ aims to find a new basis  $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_d)$  of  $L$  such that  $\|\mathbf{b}_j^*\| = \lambda_1(L_{[j,k]})$  for all indices  $j$  with  $k = \min(j+\beta-1, d)$ , by calling an SVP oracle (e.g., an enumeration algorithm in Algorithm 1) on the projected lattice  $L_{[j,k]} = \mathcal{L}(\mathbf{B}_{[j,k]})$  cyclically for  $j = 1, 2, \dots, d-1$ . During DeepBKZ reduction, the Gram-Schmidt norms  $(\|\mathbf{b}_1^*\|, \dots, \|\mathbf{b}_d^*\|)$  decrease monotonically in lexicographic order. As  $\beta$  increases, the quality of an output basis improves in both theory and practice. Similar to

---

**Algorithm 1** DeepBKZ [37]

---

```
1: procedure DeepBKZ( $\mathbf{B}$ ,  $\delta$ ,  $\beta$ )
2:    $\triangleright \mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_d)$ : basis of a lattice  $L$ ,  $\delta$ : reduction parameter,  $\beta$ : blocksize
3:    $\mathbf{B} \leftarrow \text{DeepLLL}(\mathbf{B}, \delta)$   $\triangleright$  DeepLLL-reduction for the input basis  $\mathbf{B}$ 
4:    $z \leftarrow 0, j \leftarrow 0$ 
5:   while  $z < d - 1$  do
6:      $j \leftarrow (j \bmod d - 1) + 1, k \leftarrow \min(j + \beta - 1, d), h \leftarrow \min(k + 1, d)$ 
7:      $\mathbf{v} \leftarrow \text{ENUM}(L_{[j,k]})$ 
8:      $\triangleright$  Enumeration over  $L_{[j,k]}$  to find  $\mathbf{v} \in L$  satisfying  $\|\pi_j(\mathbf{v})\| = \lambda_1(L_{[j,k]})$ 
9:     if  $\|\pi_j(\mathbf{v})\| < \|\mathbf{b}_j^*\|$  then
10:       $z \leftarrow 0, (\mathbf{b}_1, \dots, \mathbf{b}_h) \leftarrow \text{LLL}((\mathbf{b}_1, \dots, \mathbf{b}_{j-1}, \mathbf{v}, \mathbf{b}_j, \dots, \mathbf{b}_h))$ 
11:       $\triangleright$  Remove the linear dependency by LLL after insertion of  $\mathbf{v}$  at position  $j$ 
12:     else
13:        $z \leftarrow z + 1$ 
14:     end if
15:      $\text{DeepLLL}((\mathbf{b}_1, \dots, \mathbf{b}_h), \delta)$ 
16:      $\triangleright$  DeepLLL-reduction for the sub-basis  $(\mathbf{b}_1, \dots, \mathbf{b}_h)$  of the current basis  $\mathbf{B}$ 
17:   end while
18: end procedure
```

---

BKZ, when  $\beta = d$ , DeepBKZ outputs an *HKZ-reduced* basis that is the minimum among the bases of  $L$  in the lexicographic order of the Gram-Schmidt norms.

For our parallelization of DeepBKZ, we consider the lexicographic order of the Gram-Schmidt norms when comparing lattice bases. Precisely, for two sub-bases  $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_{d_1})$  and  $\mathbf{C} = (\mathbf{c}_1, \dots, \mathbf{c}_{d_2})$  of a lattice, we define an order as

$$\mathbf{B} < \mathbf{C} \iff \exists j \leq \min\{d_1, d_2\} \text{ s.t. } (\|\mathbf{b}_i^*\| = \|\mathbf{c}_i^*\|) \wedge (\|\mathbf{b}_j^*\| < \|\mathbf{c}_j^*\|) \text{ for all } i < j. \quad (2)$$

### 3.2 Strategy of parallel sharing in DeepBKZ

Our aim of parallelization is to efficiently find a small lattice basis in the lexicographic order (2) of the Gram-Schmidt norms. Our parallelization policy is a heuristic approach. Specifically, we generate a lot of different bases of a lattice through randomization. We then execute DeepBKZ on the randomized bases in parallel by sharing short lattice vectors to find a small basis in the order (2). We here denote a unit that executes DeepBKZ as a *solver*.

Given a lattice  $L$ , we state that a basis  $\mathbf{S}$  of  $L$  is *global* if it satisfies  $\mathbf{S} \leq \mathbf{B}$  in the order (2) for all bases of the solver  $\mathbf{B}$  of  $L$ . For a global basis  $\mathbf{S} = (\mathbf{s}_1, \dots, \mathbf{s}_d)$  of  $L$ , we also call its sub-basis of the form  $\mathbf{S}_k = (\mathbf{s}_1, \dots, \mathbf{s}_k)$  a *global sub-basis* for each  $1 \leq k \leq d$ . Any global sub-basis  $\mathbf{S}_k$  satisfies  $\mathbf{S}_k \leq \mathbf{B}$  for all the bases of the solver  $\mathbf{B}$  from (2). In our strategy, all solvers share a common global sub-basis  $\mathbf{S}_k$  while running DeepBKZ; in

other words, all solvers share the first  $k$  vectors of a global basis. One of the realizations of sharing the global sub-basis is message passing of the whole and a part of the basis. For the case  $k = 1$ , when the first basis vector  $\mathbf{b}_1$  of a basis  $\mathbf{B}$  is updated in a solver, the basis of that solver becomes a global basis. Thus, we set  $\mathbf{s}_1 = \mathbf{b}_1$  and send the vector is sent to all solvers. When a solver receives  $\mathbf{s}_1$ , the solver adds it to the top of its basis  $\mathbf{C} = (\mathbf{c}_1, \dots, \mathbf{c}_d)$  and performs LLL on the  $d + 1$  vectors  $(\mathbf{s}_1, \mathbf{c}_1, \dots, \mathbf{c}_d)$  to remove its linear dependency. The vector  $\mathbf{s}_1$  remains as the first basis vector in most cases; thus, we complete to share  $\mathbf{s}_1$  with the solver. If the first basis vector  $\mathbf{c}_1$  of  $\mathbf{C}$  after LLL is not equal to  $\mathbf{s}_1$ , then it must hold the  $\|\mathbf{c}_1\| \leq \|\mathbf{s}_1\|$  and the basis  $\mathbf{C}$  of the solver becomes a new global basis, using the same procedure as above is used to share  $\mathbf{c}_1$  with the other solvers. To generalize, in the case where  $k \geq 1$ , when a global basis  $\mathbf{S}$  is updated, its global sub-basis  $\mathbf{S}_k = (\mathbf{s}_1, \dots, \mathbf{s}_k)$  is sent to the other solvers, which can be merged by LLL on  $(\mathbf{s}_1, \dots, \mathbf{s}_k, \mathbf{c}_1, \dots, \mathbf{c}_d)$  and the re-sharing of a basis. In practice, it is more stable to sequentially insert one vector at a time into its basis and remove the linear dependency by LLL due to floating-point precision.

### 3.3 Implementation

We introduce new software to realize the strategy of parallel sharing DeepBKZ as described in Subsection 3.2. Our software is based on CMAP-LAP [34], a generic framework for the massive parallelization of lattice algorithms, including reduction, enumeration, and sieve algorithms. We call our software “CMAP-DeepBKZ” because it is specialized for the parallelization of DeepBKZ by using supervisor-worker style [27] functions in CMAP-LAP. In our software, we denote each worker process as *solver*. We represent the progress of a solver as the *status*, a pair consisting of a basis and a blocksize parameter for DeepBKZ. We also present a triple containing a lattice basis, algorithm parameters, and a status, called a *task*. Given an input basis of a lattice  $L$ , the supervisor process generates tasks with randomized bases of  $L$  and distributes them to solvers. The solver process executes DeepBKZ according to the received task and periodically communicates the current status to the supervisor to update or fetch a global basis while executing the reduction algorithm.

In Figure 1, we show the overall process of parallel sharing DeepBKZ in CMAP-DeepBKZ. We describe each process in Figure 1 below.

- (i) Given an input basis  $\mathbf{B}$  of a lattice  $L$ , the supervisor sets the global basis  $\mathbf{S}$  of  $\mathbf{B}$  and creates initial tasks by randomizing the lattice bases  $\mathbf{B}$ .
- (ii) The supervisor sends the tasks to idle solvers and simultaneously stores them in a solver pool.
- (iii) Every solver executes DeepBKZ according to a received task.
- (iv) Every solver sends its status  $(\mathbf{B}, \beta)$  to the supervisor periodically, where  $\mathbf{B}$  is the current reduced basis and  $\beta$  is the current blocksize of DeepBKZ.

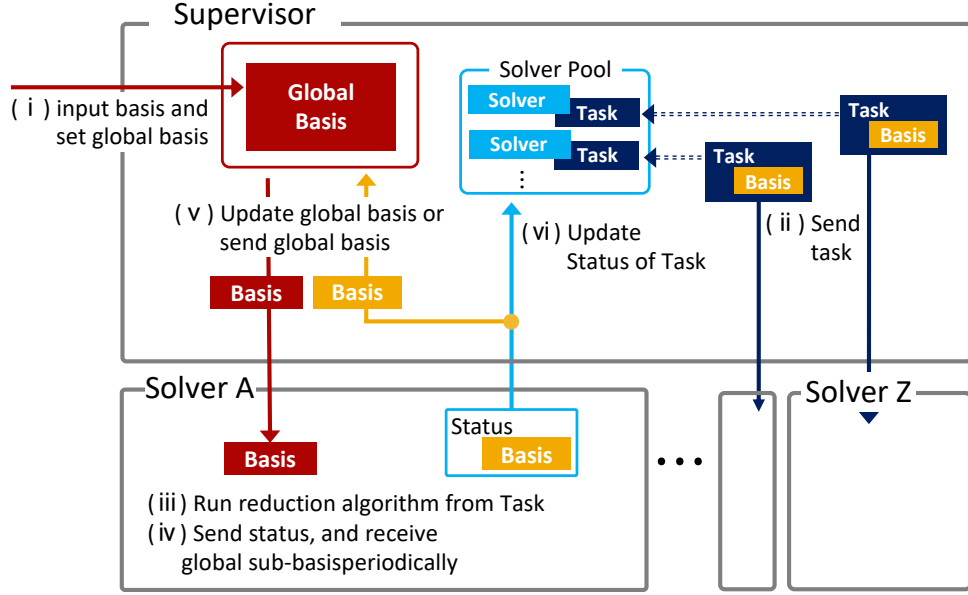


Figure 1: The overall process of parallel sharing DeepBKZ in CMAP-DeepBKZ

(v) When the supervisor receives a status  $(\mathbf{B}, \beta)$  from a solver, it compares the basis  $\mathbf{B}$  of the status with a global sub-basis  $\mathbf{S}_k$ .

- If  $\mathbf{B}$  is smaller than the global sub-basis  $\mathbf{S}_k$  in the lexicographical order of Gram-Schmidt norms, the supervisor replaces the current global basis  $\mathbf{S}$  with  $\mathbf{B}$ .
- If not, the supervisor sends the global sub-basis  $\mathbf{S}_k$  back to the solver.

(vi) The supervisor updates tasks in the solver pool according to received statuses.

Because the supervisor maintains a global basis, each solver can obtain the global basis only by communicating solely with the supervisor, that is, without communicating with other solvers. The *solver pool* maintained within the supervisor is the container of the tasks executed by solvers, which are updated according to their respective statuses sent by the solvers. The solver pool data is used to create a checkpoint file because this pool maintains the latest progress of all solvers.

### 3.3.1 Parallel framework

We have implemented CMAP-DeepBKZ based on the CMAP-LAP framework, which applies massively parallel strategies for lattice problems. The CMAP-LAP framework is designed to facilitate the implementation of other parallel strategies based on this framework. CMAP-LAP is created by inheriting from the *Generalized UG framework* (UG version 1.0 RC), a parallel framework implemented in C++11 which provides the infrastructure for supervisor-worker parallelism. The concept of Generalized UG is to parallelize the state-of-the-art solvers from the outside. Generalized UG provides several abstract classes which

can be customized according to the target problem and solvers. This customization flexibility is suitable for the realization of our strategy.

The motivation of the CMAP-LAP framework is to utilize the interactions of typical SVP algorithms such as the enumeration, sieve, and lattice reduction algorithms as *samplers* of lattice basis and vectors. For example, the lattice basis output of the lattice reduction algorithm can be used for the enumeration and sieve algorithm. Moreover, the short lattice vector found by each algorithm can accelerate the lattice basis reduction or sieve. Therefore, CMAP-LAP is designed as a scheme that can heterogeneously parallel execute solvers in parallel and share lattice vectors and bases among the solvers. It has a modular system for the implementation of new strategies relating to large-scale parallelization. Developers can customize task structures to execute multi-thread or multi-rank SVP solvers. In addition, CMAP-LAP’s communication API allows solvers to share information synchronously, quickly, and safely with minimal changes. Furthermore, CMAP-LAP has a flexible and high-level checkpointing function. Thereby, we can challenge to solve high-dimensional SVP instances which require millions of core hours. In [34], the stability and future performance of the framework are shown by the several experiments of heterogeneous and long-running execution of the naive algorithms combinations in a large-scale environment using up to 103,680 cores.

### 3.3.2 Processing flow of the supervisor and solver

The pseudo processing flow in the supervisor of CMAP-DeepBKZ is shown in Algorithm 2. The supervisor continuously checks whether it has received a message from the solvers using the `MPI_Iprobe` function. If messages have been sent to the supervisor, it handles the received message according to its tag, which represents the message type. In CMAP-DeepBKZ, the most important and frequently exchanged message tag is *TagSolverState*, which indicates the status of the algorithm. The status is a pair consisting of the basis and the blocksize of DeepBKZ. In CMAP-LAP, the supervisor only uses the status to update the tasks in the solver pool. In addition, the supervisor in CMAP-DeepBKZ updates and distributes the global basis  $\mathbf{S}$  using basis  $\mathbf{B}$  of the status. If  $\mathbf{B} < \mathbf{S}_k$ , that is  $\mathbf{B}$  satisfies the condition to be the global basis, the supervisor then updates the global basis  $\mathbf{S}$  to  $\mathbf{B}$ . If  $\mathbf{S}_k < \mathbf{B}$ , the supervisor sends the global sub-basis  $\mathbf{S}_k$  back to the solver. Because the CMAP-DeepBKZ has this supervisor-worker style, we can share the global-sub basis using this simple process.

The algorithmic function executed by the solver is shown in Algorithm 3. The solver communicates using the communication API in CMAP-LAP. It periodically sends the basis and the currently running blocksize as status until the algorithm terminates. If the solver’s basis is smaller than the global sub-basis, the solver receives the global sub-basis from the supervisor. As shown in Algorithm 3, almost any algorithm can be applied to our software because we are only required to customize for the communication between subroutines. The experiment of running several algorithms in parallel is described in [34].

### 3.3.3 Checkpoint and Restart

It is critical to save the progress of the solvers to resume the computation because it takes a significantly large amount of core hours to solve the large-dimension SVPs. This is accomplished by powerful checkpointing functionality in **C**MAP-DeepBKZ that stores the complete progress information of the SVP solvers. Because the supervisor periodically receives the algorithm’s progress from these solvers, it tracks progress and writes it to the checkpoint file. More specifically, whenever the supervisor receives a status from the solver, it updates the task in the solver pool based on the received status. When a checkpoint is requested, the supervisor serializes the tasks data in the solver pool, compresses and writes them by using zlib [13], a portable compression library. When we resume the computation, the tasks are loaded from the checkpoint file and stored in the *task pool*, a container of tasks waiting for execution. Next, the supervisor distributes the tasks to solvers according to the priority associated with the tasks. The supervisor creates new tasks when many solvers are available. In contrast, if the number of solvers is less than that when the checkpoint file was generated, the tasks remain in the task pool and are given priority when the supervisor distributes the next task.

## 4 Similarity of lattice bases

The benefit of parallelization in our algorithm mainly depends on the randomization of bases. Each solver works independently on a randomized copy of the input basis, and we hope that the reduction algorithm works faster for a certain random copy. While this independence allows for asynchronous parallelization, the overall system would benefit from collaboration among solvers. Therefore, we introduced a sharing scheme in the previous section in which solvers indirectly exchange short lattice vectors with each other via the supervisor. However, there is a trade-off between randomness and the amount of shared information. Let us think of the extreme case when all vectors are shared and all solvers work on the same basis, the benefit of parallelization would be completely nullified. It is important to ensure that the diversity of the bases is preserved by the sharing. In this section, we introduce a novel metric to quantify the diversity of lattice bases. This metric will be used to determine the value of the number of share vectors  $k$  for the optimal balance.

### 4.1 Grassmann metrics

Let  $\mathbf{B}$  and  $\mathbf{C}$  be two bases of a  $d$ -dimensional lattice in  $\mathbb{R}^n$ . We define several similarity metrics between  $\mathbf{B}$  and  $\mathbf{C}$ , and use them to quantify the diversity of a set of bases. Recall that DeepBKZ with blocksize  $\beta$  is an algorithm to find a basis whose  $i$ -th basis vector  $\mathbf{b}_i$  which is the shortest from the projected lattice  $\mathcal{L}(\mathbf{B}_{[i, \min(i+\beta-1, d)]})$  for all  $i$ . It is natural to compare the projected lattices  $\tilde{B}_i := \mathcal{L}(\mathbf{B}_{[i, d]})$  and  $\tilde{C}_i := \mathcal{L}(\mathbf{C}_{[i, d]})$  for each  $1 \leq i \leq d$ . Each  $\tilde{B}_i$  defines an  $m$ -dimensional subspace in  $\mathbb{R}^n$ , where  $m = d - i + 1$ . The subspace corresponds to a point in the *Grassmannian manifold*  $Gr(m, n)$  which consists of  $m$ -dimensional linear

subspaces in the Euclidean space  $\mathbb{R}^n$ . The Grassmannian manifold comes equipped with several metrics (distances), which we use as the similarity measures for  $\tilde{B}_i$  and  $\tilde{C}_i$ .

Let  $Y^i(\mathbf{B})$  be the  $(m \times n)$ -orthonormal matrix corresponding to  $\tilde{B}_i$  whose rows are  $\mathbf{b}_k^*/\|\mathbf{b}_k^*\|$  for  $i+1 \leq k \leq d$ . The standard way to define metrics on the Grassmannian manifold is via principal angles [5]. Denote the singular value decomposition (SVD) of  $Y^i(\mathbf{B})Y^i(\mathbf{C})^T$  by

$$Y^i(\mathbf{B})Y^i(\mathbf{C})^T = U \text{diag}(\cos \theta_1, \dots, \cos \theta_m) V, \quad (3)$$

where  $U$  and  $V$  are orthonormal matrices and the singular values  $\cos \theta_k$  are sorted in decreasing order. The singular values  $\cos \theta_k$  are called *canonical correlations* and the angles  $\theta_1, \dots, \theta_m \in [0, \pi/2]$  are called the *principal angles* of  $Y^i(\mathbf{B})$  and  $Y^i(\mathbf{C})$  [6, 19]. The first principal angle  $\theta_1$  is the minimal angle between the two subspaces spanned by  $\tilde{B}_i$  and  $\tilde{C}_i$ . If this minimal angle is achieved by  $u_1 \in \text{Span}(\tilde{B}_i)$  and  $v_1 \in \text{Span}(\tilde{C}_i)$ , the second principal angle  $\theta_2$  is the minimal angle between their orthogonal complements. The third and subsequent principal angles are defined in a similar manner.

The *geodesic distance*, which is induced by the canonical Riemannian metric on  $Gr(m, n)$  as the homogeneous space of the orthogonal group  $O(n)$ , is computed as  $d(Y^i(\mathbf{B}), Y^i(\mathbf{C})) = \sqrt{\sum_i \theta_i^2}$ . Although the geodesic distance is the most natural and “authentic” metric on  $Gr(m, n)$ , it is computationally expensive since we have to compute the SVD of a large matrix. It is thus preferable to use metrics that can be computed efficiently without invoking SVD. Such metrics include *chordal metric*  $d_c$  and the *projection 2-norm metric*  $d_{p2}$ . The *chordal metric* is defined as the square root of the square sum of the sine of principal angles, but can be computed efficiently by the Frobenius norm of the difference of the projectors:

$$d_c(Y^i(\mathbf{B}), Y^i(\mathbf{C})) := \sqrt{\sum_k \sin^2 \theta_k} = \frac{1}{\sqrt{2}} \|Y^i(\mathbf{B})^T Y^i(\mathbf{B}) - Y^i(\mathbf{C})^T Y^i(\mathbf{C})\|_F.$$

It is shown in [16, Section 4.3] that the chordal metric provides a lower bound, and in fact, a good approximation to the geodesic distance.

The maximum principal angle  $\theta_m$  is a generalization of the dihedral angle between two planes in  $\mathbb{R}^3$ , and hence, it is another natural metric to measure the diversity of bases. The *projection 2-norm metric* is defined as

$$d_{p2}(Y^i(\mathbf{B}), Y^i(\mathbf{C})) := \sin \theta_m = \|Y^i(\mathbf{B})^T Y^i(\mathbf{B}) - Y^i(\mathbf{C})^T Y^i(\mathbf{C})\|_2.$$

Note that the largest singular value can be efficiently computed by the power method. When  $n$  is sufficiently large, the maximum principal angle for random  $\mathbf{B}$  and  $\mathbf{C}$  is close to  $\pi/2$ , and the projection 2-norm metric is closed to one regardless of  $i$ .

## 4.2 Diversity of bases

Given a multiset  $\mathcal{B} = (\mathbf{B}_1, \dots, \mathbf{B}_m)$  of lattice bases, we define its diversity using the Grassmann metrics defined in the previous subsection.

**Definition 1 (Diversity of projected lattices)** Let  $P(\mathcal{B})$  be the set of all pairs of elements in  $\mathcal{B}$ . We define its  $i$ -th projected diversity associated to a Grassmann metric  $d_g$  as the mean of the pairwise distance:

$$\text{Div}^i(\mathcal{B}, d_g) := \frac{1}{|P(\mathcal{B})|} \sum_{(\mathbf{B}, \mathbf{C}) \in P(\mathcal{B})} d_g(Y^i(\mathbf{B}), Y^i(\mathbf{C})).$$

The total projected diversity is defined by the mean of the  $i$ -th projected diversity for  $1 \leq i \leq d$ :

$$\text{Div}(\mathcal{B}, d_g) := \frac{1}{d} \sum_{i=1}^d \text{Div}^i(\mathcal{B}, d_g).$$

The higher value of  $\text{Div}(\mathcal{B}, d_g)$  indicates the greater diversity of the bases.

### 4.3 Effect of sharing short vectors on the diversity of bases

Here, we investigate how the diversity of the bases is affected by our sharing scheme. To set up a controlled experiment, we run the parallel DeepBKZ in a synchronous manner. Initially, each solver receives a randomized copy of the input lattice basis. Each iteration starts by running a tour of DeepBKZ. The global basis is defined as the minimum among all the bases of the solvers in terms of the order defined in (2). All solvers share the top- $k$  lattice vectors of the global basis as shown in lines 10–16 of Algorithm 3. The diversity  $\text{Div}(\mathcal{B}, d_g)$  is computed at this point. We then repeat the iteration.

We set the number of solvers  $m = 100$  and DeepBKZ blocksize  $\beta = 30$ , and perform this experiment with various numbers of shared vectors  $k \in \{0, 1, 8, 16, 32, 64, 80\}$  for five 90-dimensional instances of the SVP challenge.

**Snapshot of the diversity** Figure 2 shows the snapshot of  $\text{Div}^i(\mathcal{B}, d_g)$  averaged for the five SVP instances after 100 tours of DeepBKZ. We observe that the shapes of  $\text{Div}^i(\mathcal{B}, d_g)$  and  $\text{Div}^i(\mathcal{B}, d_c)$  are almost identical up to scaling, as  $d_c$  gives a good approximation to  $d_g$ . In the following analysis, we will focus on  $d_c$  and  $d_{p2}$  as they can be efficiently computed. Note that  $\text{Div}^i(\mathcal{B}, d_g)$  for  $i \leq k$  are not necessarily 0 although we share the top- $k$  vectors of the global basis. This is because the top- $k$  vectors of the basis of each solver are updated by the insertion and LLL. We observe that the values  $\text{Div}^i(\mathcal{B}, d_g)$  decrease as the number of shares  $k$  increases. This is an expected result in agreement with our intuition, and it is implied that the diversity of the projected lattice can be quantified by the proposed  $i$ -th projected diversity metric. When  $k = 0$  and there is no sharing, the shape for the chordal metric shows a symmetry with respect to  $i = 45$ . This is due to the one-to-one correspondence between  $Gr(m, n)$  and  $Gr(n - m, n)$  that maps an  $m$ -dimensional subspace to its orthogonal complement. The deviation of the shape of  $\text{Div}^i(\mathcal{B}, d_g)$  from that for the  $k = 0$  case indicates the decrease in the diversity of the bases due to the sharing. We indeed observe that the shape is closer to that of  $k = 0$  for smaller  $k$ 's.

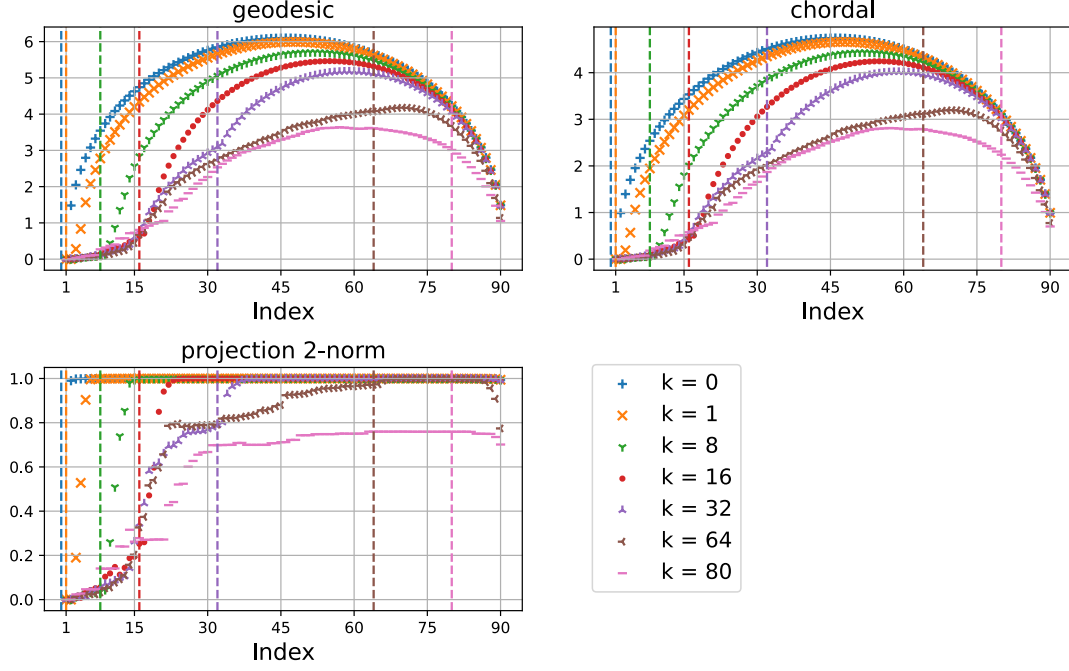


Figure 2: The average of the  $i$ -th projected diversity  $\text{Div}^i(\mathcal{B}, d_g)$  computed for 90-dimensional lattice bases with different numbers of shared vectors  $k$  right after 100 DeepBKZ tours.)

For the projection 2-norm,  $\text{Div}^i(\mathcal{B}, d_{p2})$  is close to one for all  $i$  when  $k = 0$ , as expected. This ensures that each solver works on a different search space. When  $k = 64$  and  $80$ , the lower values of  $\text{Div}^i(\mathcal{B}, d_{p2})$  suggest that there is substantial overlap among the search spaces of the solvers, which would affect the overall efficiency of the system. We will discuss this point later in a large-scale experiment in Section 5.4.

**Transition of the total diversity with tours** Figure 3 shows the transition of the total diversity  $\text{Div}(\mathcal{B}, d_g)$  with respect to the number of tours. We observe that when  $k = 0$ , the total diversity stays constant, indicating that the DeepBKZ algorithm preserves the diversity of bases and the lattice reduction itself does not reduce the diversity of bases. We observe that when  $k > 0$ , the total diversity decreases at the early stage and then converges to a certain value which depends on  $k$ . This experiment shows that the diversity of the bases of the solvers is preserved to some extent during the execution of our shared DeepBKZ algorithm, even though the randomization is performed only once before the first tour. We confirm these observations through a large-scale experiment in Section 5.4.

**Evaluation of different randomization** Our novel diversity metric has the potential to be applied for various analyses of a set of lattice bases. For example, we conduct an evaluation of the effect of different randomization methods. In general, the quality of

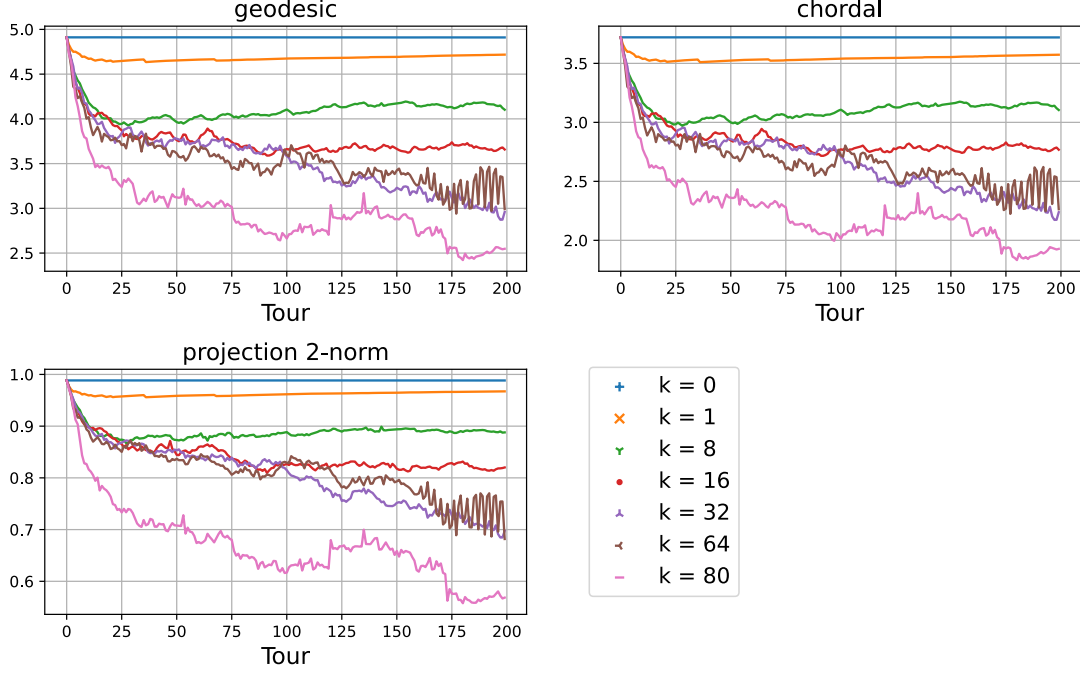


Figure 3: Transition of the total diversity  $\text{Div}(\mathcal{B}, d_g)$  computed for 90-dimensional lattice bases with different numbers of shared vectors  $k$  after each tour of DeepBKZ.

the random element generator has a large impact on the performance of a randomized algorithm. In our case, the input basis is multiplied by randomly generated unimodular matrices to produce different bases for the input lattice. We compare three popular ways to generate unimodular matrices using our diversity metric.

- *LU*: A pair consisting of a lower and an upper integer triangular matrix with 1's along the diagonal is generated. They are then multiplied after their rows are randomly shuffled.
- *Swap*: A permutation matrix is generated uniformly randomly.
- *Fplll*: A permutation matrix is generated uniformly randomly. Then, row operations are performed on it three times, picking a row to add to or subtract from another row. This is used by the `fplll` library.

First, we generate 100 bases from a single lattice basis by one of the above methods. Then, we calculate  $\text{Div}^i(\mathcal{B}, d_g)$  after (i) randomization, (ii) randomization and LLL, and (iii) randomization and a tour of DeepBKZ without sharing. Figure. 4 shows the average of  $\text{Div}^i(\mathcal{B}, d_g)$  of five 90-dimensional instances of the SVP challenge. The three lines corresponding to the three methods grow closer as one proceeds from (i) to (iii). This implies that the three methods are all exhibit bias, but this bias is eliminated by LLL and DeepBKZ. Therefore, in practice, it is not necessary to pay significant attention to the

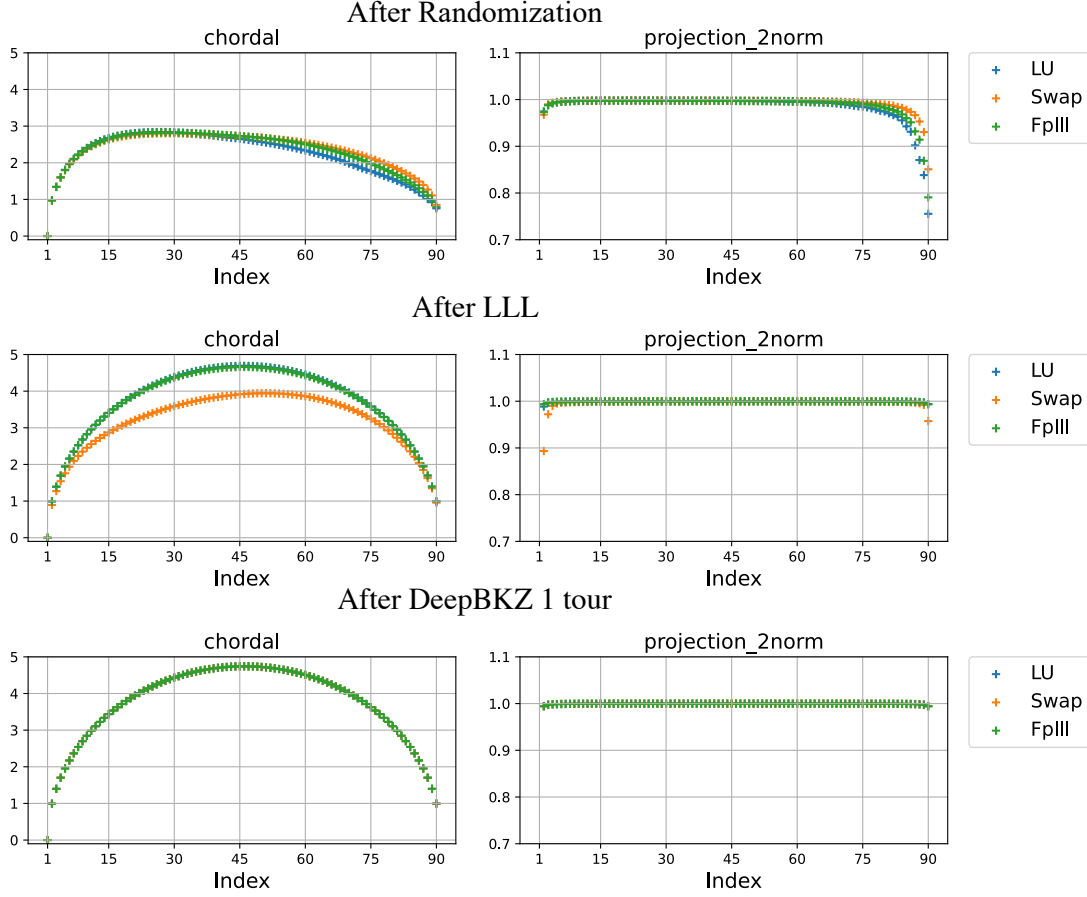


Figure 4: The  $i$ -th projected diversity for the chordal (left) and the projection 2-norm (right) Grassmann metrics computed (top) immediately after randomization, (middle) after LLL, and (bottom) after one tour of DeepBKZ for 90-dimensional lattice bases with different random generation models of unimodular matrices.

randomization method. It is interesting that the reduction process itself contributes to the diversity of the bases.

**Remark 1 (Distribution of reduced bases)** *Some lattice algorithms assume the randomness of input bases. For example, extreme pruning [18], a pruning technique for enumeration, relies on the heuristic of [18, Heuristic 3] that the normalized Gram-Schmidt vectors  $(\mathbf{b}_1^*/\|\mathbf{b}_1^*\|, \dots, \mathbf{b}_d^*/\|\mathbf{b}_d^*\|)$  of a basis is uniformly distributed. This heuristic allows us to estimate the probability that a vector of a given length is included in a pruned enumeration tree. However, to our best knowledge, this heuristic has not yet been verified in detail for reduced bases, or more precisely, bases obtained by a reduction algorithm. Below, we apply our diversity metric to provide supportive evidence for [18, Heuristic 3].*

*Note that we can sample uniformly from  $\text{Gr}(1, n)$  by sampling from the  $n$ -dimensional normal distribution with the zero mean and the identity covariance. By sampling  $m$  ele-*

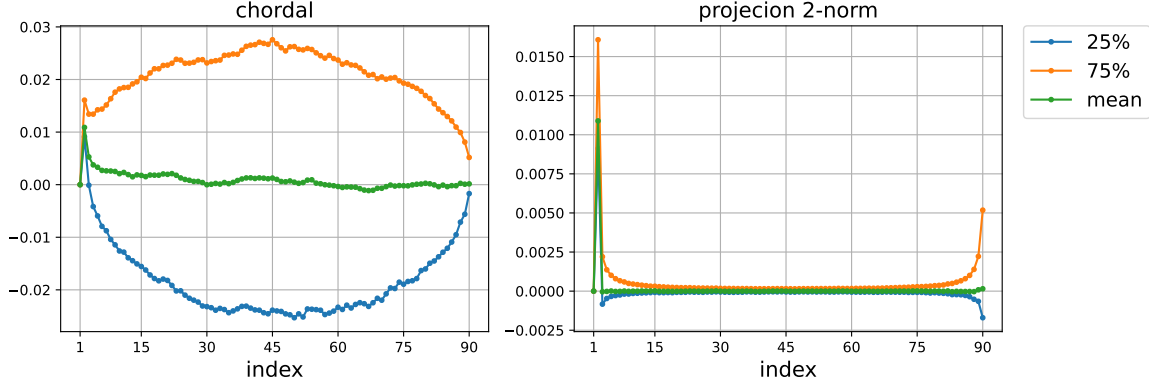


Figure 5: Comparison between the diversity metrics of  $\mathcal{C}$  and that of  $\mathcal{B}$ . mean:  $(i, \text{Div}^i(\mathcal{C}, d_g) - \text{Div}^i(\mathcal{B}, d_g))$ , 25%:  $(i, \text{Div}_{25\%}^i(\mathcal{C}, d_g) - \text{Div}^i(\mathcal{B}, d_g))$ , 75%:  $(i, \text{Div}_{75\%}^i(\mathcal{C}, d_g) - \text{Div}^i(\mathcal{B}, d_g))$  for (Left)  $d_g = d_c$  the chordal metric, and (Right)  $d_g = d_{p2}$  the projection 2-norm.

ments independently from  $\text{Gr}(1, n)$ , we obtain an element of  $\text{Gr}(m, n)$  almost surely. Let  $\mathcal{C}_i$  be a multiset of elements in  $\text{Gr}(d - i + 1, n)$  sampled in this manner. The value

$$\text{Div}^i(\mathcal{C}_i, d_g) := \frac{1}{|P(\mathcal{C}_i)|} \sum_{(\mathbf{B}, \mathbf{C}) \in P(\mathcal{C}_i)} d_g(\mathbf{B}, \mathbf{C})$$

represents the diversity of randomly sampled subspaces. We compare this value with the  $i$ -th projected diversity of the subspaces defined by the lattice bases derived from a single lattice basis by the randomization and the DeepBKZ algorithm. If the distribution of the reduced bases is similar to that of random bases, the diversity metrics of these two groups should be similar. As in Section 4.3, we generate a multiset of the lattice bases  $\mathcal{B}$  by running DeepBKZ for 100 tours with no sharing ( $k = 0$ ) on 100 random copies (generated by the LU method) of a single instance of 90-dimensional SVP challenge. Figure 5 shows the difference between  $\text{Div}^i(\mathcal{B}, d_g)$  and  $\text{Div}^i(\mathcal{C}_i, d_g)$ , where  $d = 90$  and  $|\mathcal{C}_i| = 100$ . In addition to the difference of means, the difference between  $\text{Div}^i(\mathcal{B}, d_g)$  and the quartiles, denoted by  $\text{Div}_{25\%}^i(\mathcal{C}, d_g)$  and  $\text{Div}_{75\%}^i(\mathcal{C}, d_g)$ , of  $\mathcal{C}_{d_g}^i := \{d_g(Y^i(\mathbf{B}), Y^i(\mathbf{C})); (\mathbf{B}, \mathbf{C}) \in P(\mathcal{C})\}$  are shown. We observe that the difference between  $\text{Div}^i(\mathcal{B}, d_g)$  and  $\text{Div}^i(\mathcal{C}_i, d_g)$  is close to zero. In fact,  $\text{Div}^i(\mathcal{B}, d_g)$  fall within the quartiles of the diversity of the random elements  $\mathcal{C}_{d_g}^i$  except for  $i = 2$ . The same is observed for other four instances of 90-dimensional SVP challenge. Note that when  $i = 2$ , the first basis vector  $\mathbf{b}_1$  is likely to be the shortest vector, and hence, reduced bases share the same first basis vector with a certain probability.

This result suggests that the assumption of [18, Heuristic 3] holds for bases reduced by DeepBKZ except for the first vector.

Table 1: Computing platforms, operating systems, compilers and libraries

Machine	Memory / node	CPU	CPU frequency	# nodes	# cores
Lisa	384 GB	Xeon Platinum 9242	2.30 GHz	1,080	103,680
Emmy	384 GB	Xeon Platinum 9242	2.30 GHz	128	12,288
ITO	192 GB	Xeon Gold 6154	3.00 GHz	128	4,608
CAL A	256 GB	Xeon E5-2640 v3	2.60 GHz	4	64
	256 GB	Xeon E5-2650 v3	2.30 GHz	4	80
CAL C	32 GB	Xeon E3-1284L v3	1.80 GHz	45	180

*Operating systems and versions:* Lisa and Emmy [CentOS Linux release 7.7.1908], ITO [Red Hat Enterprise Linux Server release 7.3.1611], CAL A and CAL C [CentOS Linux release 7.9.2009]. *Compilers and versions:* Lisa and Emmy [intel19.0.5, impi2019.5], ITO [icc 19.1.1.217, impi2019.4], CAL A [icc 19.1.3.304, openmpi4.0.5], CAL C [icc19.1.3.304, impi2020.4.304]. *Libraries and versions:* NTL v11.3.3, Eigen v3.3.7, gsl v2.6, OpenBLAS v0.3.7, fplll v5.2.1.

## 5 Numerical experiments

In this section, we show experimental results to demonstrate the performance of CMAP-DeepBKZ in a large-scale computing environment. We used the computing platforms in Table 1 and conducted experiments using up to 103,680 cores. The supercomputers Lisa and Emmy are in the HLRN IV system at Zuse Institute Berlin, and the ITO supercomputer is at Kyushu University. The CPU cluster computers CAL A and CAL C possess a total of 144 and 180 cores, respectively. We used MPI processes without hyper-threading. For our experiments, we used instances in the Darmstadt SVP challenge [28], but we reduced every instance in advance using LLL implemented in the fplll library [36].

### 5.1 Metrics to measure the output quality of reduction algorithms

As described below, we present typical metrics to measure the output quality of reduction algorithms to compare the experimental results later.

- *Hermite factor:* Let  $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_d)$  be a basis of a lattice  $L$  output by a reduction algorithm. Assume that  $\mathbf{b}_1$  is the shortest among the vectors  $\mathbf{b}_i$ 's. Then, the *Hermite factor* of the reduction algorithm is defined as  $\gamma = \frac{\|\mathbf{b}_1\|}{\text{vol}(L)^{1/d}}$ . As  $\gamma$  is smaller, a reduction algorithm can find a shorter basis vector. Exhaustive experiments in [17] show that for a practical reduction algorithm such as LLL and BKZ, the root Hermite factor  $\gamma^{1/d}$  converges to a constant value for high dimensions  $d \geq 100$ . Therefore, the root Hermite factor  $\gamma^{1/d}$  is a useful metric to compare the identical output quality of practical reduction algorithms for lattice bases in high dimensions.

- *Enumeration Cost*: Given a basis  $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_d)$  of a lattice  $L$ , we can estimate the cost to find a shortest non-zero vector in  $L$  by enumeration using  $\mathbf{B}$ . Given a search radius  $R > 0$ , an enumeration tree of depth  $n$  is constructed whose nodes at depth  $d - k + 1$  correspond to the set of all vectors in  $\pi_k(L)$  with a maximum length of  $R$ . The key observation here is that if a shortest vector  $\mathbf{s}$  satisfies  $\|\mathbf{s}\| \leq R$ , its projections must also satisfy  $\|\pi_k(\mathbf{s})\|^2 \leq R^2$  for all  $1 \leq k \leq d$ . Hence, it appears as a leaf of the tree. These  $d$  inequalities provide an enumeration of the tree. The total number of nodes to be traversed is estimated using the Gaussian Heuristic as  $N = \sum_{k=1}^d H_k$ , where  $H_k = \frac{R^k \omega_k}{\text{vol}(\pi_{d+1-k}(L))}$  for every  $1 \leq k \leq d$  (see [18] for details). As  $\mathbf{B}$  is reduced, the total number of nodes  $N$  decreases in practice.
- *Geometric Series Assumption (GSA)*: Let  $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_d)$  be a reduced basis, and  $\mathbf{b}_1^*, \dots, \mathbf{b}_d^*$  its Gram-Schmidt vectors. The GSA in [30] states that the plots of log-norms  $\log \|\mathbf{b}_i^*\|$  of Gram-Schmidt vectors approximate a straight line. (For a  $\beta$ -BKZ-reduced basis, the GSA does not hold for the last  $d - \beta$  plots because the last block  $\mathbf{B}_{[d-\beta+1:d]}$  is HKZ-reduced. See [2, Figure 1] for an example of the GSA and its tail-adapted version.) To measure the average quality of  $\mathbf{B}$ , `fpylll` [36] adopts a least squares fit of  $\log \|\mathbf{b}_i^*\|^2$  for  $1 \leq i \leq d$  is adopted in `fpylll` [36] as a slope metric  $\rho$ . Under the GSA, the slope relates to the root Hermite factor via  $\gamma^{1/d} = \exp(-\frac{\rho}{4})$ .

## 5.2 Efficacy when sharing short lattice vectors

Here, we demonstrate the efficacy of CMAP-DeepBKZ when sharing short lattice vectors among solvers.

### 5.2.1 Analysis using deterministic parallel execution

First, we conducted experiments to accurately evaluate the effect of sharing short lattice vectors for 95, 100, and 105-dimensional SVP. We used the parallel DeepBKZ in the synchronous manner described in Section 4.3. By repeatedly running a tour of DeepBKZ, sharing, and distributing the global basis for each step, the behaviors of the solvers become deterministic. By contrast, in CMAP-DeepBKZ, a global basis is updated and distributed asynchronously through MPI communication. It is difficult to completely control the shared lattice vectors using CMAP-DeepBKZ.

We executed the parallel DeepBKZ with the number of solvers set to  $m = 128$ , while changing the number of short vectors shared among the solvers. In particular, we used  $k = 0, 2, 4, 8, 16, 32$ , and  $64$  as the number of short lattice vectors shared among the solvers, and we performed 10 runs for each value of  $k$ . (The case where  $k = 0$  means that no vector is shared among the solvers.) The initial blocksize of DeepBKZ is set as  $\beta = 30$ , and execution times are adjusted according to the dimension of the SVP instances. In Figure 6, we show the transition of the averages of minimum root Hermite factors, enumeration costs, and GSA slopes when running our parallel DeepBKZ. (For the enumeration cost, we set

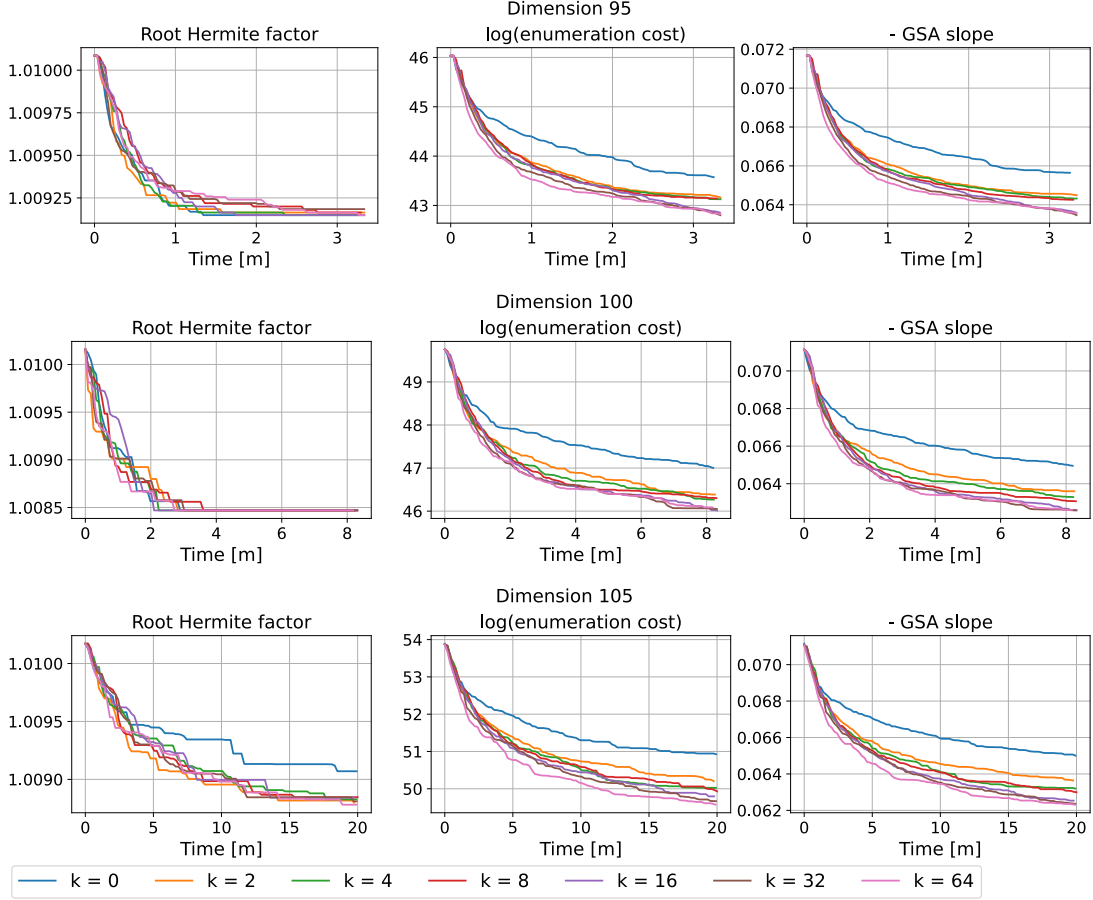


Figure 6: Transition of metrics on the output quality of parallel sharing DeepBKZ in dimension  $d = 95$  (Top), 100 (Middle) and 105 (Bottom), by using  $k = 0, 2, 4, 8, 16, 32$  and 64 as the number of short vectors shared among solvers using (Left: the average root Hermite factor  $\gamma^{1/d}$ , Center: the logarithm of the average enumeration cost  $\log(N)$ , Right: the minus of the average GSA slope  $-\rho > 0$ )

$R = \text{GH}(L)$  as the search radius of enumeration.) Comparing the results for  $k = 0$  and  $k > 0$ , we see that enumeration costs and GSA slopes  $\rho$  decreased when sharing short lattice vectors. This means that more reduced bases can be obtained through the sharing of short lattice vectors. However, the root Hermite factor transitions in dimensions  $d = 95$  and 100 were not explicitly different for the various value of  $k$ , and variation appeared only in dimension  $d = 105$ . This result shows that for 95-dimensional and 100-dimensional SVP, DeepBKZ with a blocksize  $\beta = 30$  could find shortest vectors by only utilizing the effect of parallel lattice reductions through randomization. In contrast, for SVPs of dimensions  $d \geq 105$ , parallel lattice reduction by randomization was insufficient. This finding implies that the transition of the root Hermite factor, enumeration costs, and GSA slopes can be reduced by speeding up DeepBKZ through short vector sharing, in exchange for some loss

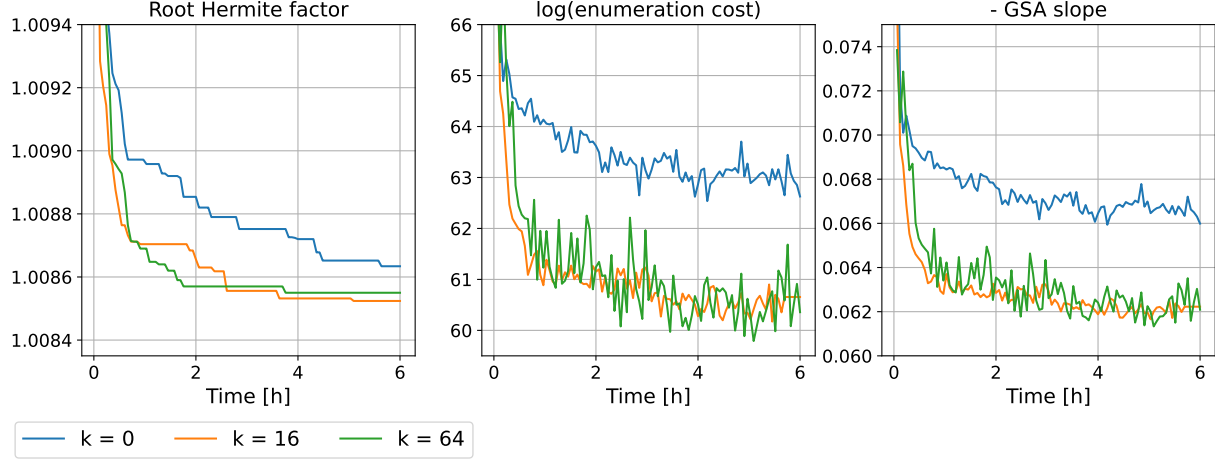


Figure 7: Same as Figure 6, but using CMAP-DeepBKZ and dimension  $d = 118$ , by using  $k = 0, 16$  and  $64$  as the number of short vectors shared among solvers

of basis diversity in a few dimensions.

### 5.2.2 Analysis of MPI parallelization using CMAP-DeepBKZ

In Figure 7 and Table 2, we display the experimental results of CMAP-DeepBKZ for the instances of the Darmstadt SVP challenge in dimension  $d = 118$  with seeds ranging from 2 to 6. Specifically, we used  $k = 0, 16$ , and  $64$  as the number of short lattice vectors shared among the solvers. We ran CMAP-DeepBKZ for six hours for each SVP instance on the supercomputer system ITO using 2,304 cores (see Table 1 for ITO). Each solver ran DeepBKZ with a blocksize  $\beta = 30$  and sent the current status to the supervisor at an interval of 120 seconds. (In other words, each solver obtained a global sub-basis of size  $k$  every 120 seconds.) In Figure 7, we show the transition of the averages of global basis's root Hermite factors, enumeration costs, and GSA slopes when running CMAP-DeepBKZ. In Table 2, we summarize the experimental results of CMAP-DeepBKZ after six hours of execution. As illustrated in Figure 7 and Table 2, it is effective to share short lattice vectors to decrease the metrics of DeepBKZ for finding short lattice vectors. For example, the minimum of the logarithm of the enumeration cost is 62.6578 (resp., 59.7701) for  $k = 0$  (resp.,  $k = 64$ ) as shown in Figure 7, and we calculate  $e^{59.7701}/e^{62.6578} \approx 0.0557$ . This implies that enumeration costs can be reduced by 5.57%, through sharing 64 short lattice vectors among the solvers.

**Remark 2 (Comparison with BKZ)** *In cryptanalysis, BKZ and its variants such as BKZ 2.0 [11] are de facto standard reduction algorithms utilized to evaluate the security of lattice-based cryptography (see [4] for details). Under the GSA and the Gaussian Heuristic, a limiting value of the root Hermite factor of BKZ with blocksize  $\beta$  for a  $d$ -dimensional*

Table 2: Experimental results of CMAP-DeepBKZ after 6 hours execution for instances of the Darmstadt SVP challenge in dimension  $d = 118$  with seeds 2–6 ( $k$  denotes the number of short vectors shared among solvers, and  $\mathbf{b}_1$  the shortest basis vector of all solver’s bases)

SVP instance	Number of shares	Updated time [h]	Norm of $\mathbf{b}_1$	Approx. factor $\frac{\ \mathbf{b}_1\ }{\text{GH}(L)}$	Root Hermite factor $\gamma^{1/d}$	Machine (Table 1)
seed2	$k = 0$	4.4354	2818.92	1.0272	1.00867	ITO
seed3		4.4358	2785.57	1.0117	1.00854	
seed4		2.2824	2834.39	1.0308	1.00870	
seed5		4.3073	2787.56	1.0153	1.00857	
seed6		5.5766	2837.97	1.0303	1.00869	
Average				1.0231	1.00863	
seed2	$k = 16$	2.0172	2789.09	1.0163	1.00858	ITO
seed3		3.6039	2770.70	1.0063	1.00849	
seed4		0.8736	2793.29	1.0159	1.00857	
seed5		5.0591	2764.17	1.0068	1.00850	
seed6		2.5595	2768.58	1.0051	1.00848	
Average				1.0101	1.00852	
seed2	$k = 64$	1.7197	2789.09	1.0163	1.00858	ITO
seed3		1.5907	2785.57	1.0117	1.00854	
seed4		1.2151	2799.01	1.0179	1.00859	
seed5		1.0780	2765.60	1.0073	1.00850	
seed6		3.7370	2786.96	1.0118	1.00854	
Average				1.0130	1.00855	

lattice is predicted in [10] as

$$\lim_{d \rightarrow \infty} \gamma^{\frac{1}{d}} = \left( \omega_{\beta}^{-\frac{1}{\beta}} \right)^{\frac{1}{\beta-1}} \sim \left( \frac{\beta}{2\pi e} (\pi\beta)^{\frac{1}{\beta}} \right)^{\frac{1}{2(\beta-1)}} \quad (4)$$

for  $\beta > 50$  and  $\beta \ll d$  (see [10, 11, 41] for experimental results supporting the prediction). Table 2 shows that CMAP-DeepBKZ can achieve the root Hermite factor around  $\gamma^{1/d} = 1.0085$  with average by blocksize  $\beta = 30$  in dimension  $d = 118$ . (See also Tables 3, 5 and 6 for root Hermite factors of CMAP-DeepBKZ in other dimensions.) In contrast, the prediction (4) implies that BKZ requires around  $\beta = 115$  to achieve the same root Hermite factor. Recall that it is the most dominant factor in both BKZ and DeepBKZ to run an exact-SVP algorithm over projected lattices of dimension  $\beta$ , and the cost is  $2^{O(\beta^2)}$  when using an enumeration algorithm for solving exact-SVP in dimension  $\beta$ . Therefore, CMAP-DeepBKZ is significantly more efficient than BKZ without parallelization.

**History of updating global bases** In Figure 8, we display the history of updating a global basis when running CMAP-DeepBKZ with the number of shares  $k = 16$  for the SVP instance in dimension  $d = 118$  with seed 5, which is the result of updating the shortest vector at the latest time. Each plot  $(x, y)$  in the figure indicates that a global

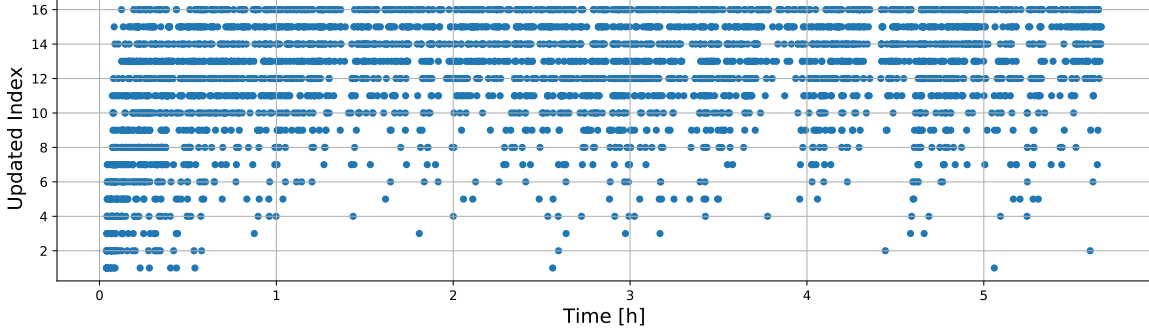


Figure 8: History of updating a global basis in an execution of CMAP-DeepBKZ with the number of shares  $k = 16$  in dimension  $d = 118$  (Each plot  $(x, y)$  indicates that a global basis at index  $y$  was updated at time  $x$ )

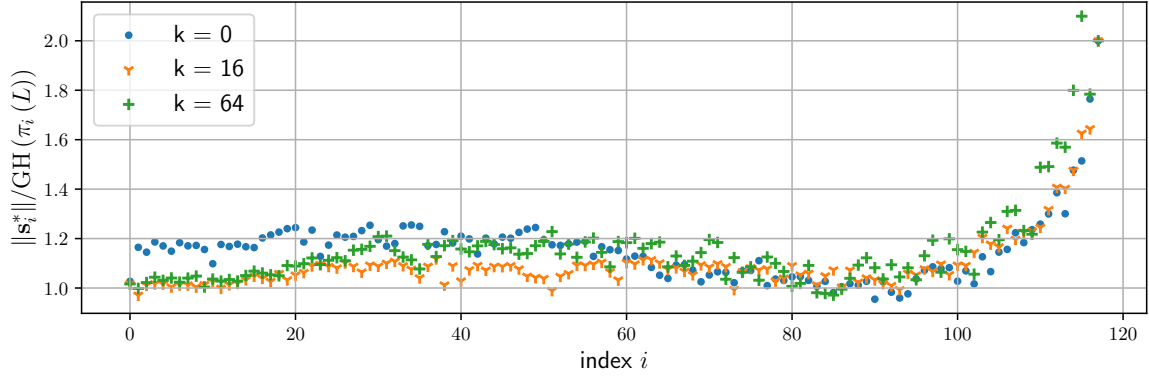


Figure 9: Plots of approximation factors in projected lattices  $\|s_i^*\|/\text{GH}(\pi_i(L))$  for a global basis  $\mathbf{S} = (s_1, \dots, s_d)$  of a lattice  $L$  of dimension  $d = 118$ , output by CMAP-DeepBKZ after 6 hours execution (We used  $k = 0, 16$  and  $64$  as the number of shares in CMAP-DeepBKZ)

basis  $\mathbf{S} = (s_1, \dots, s_d)$  at index  $y$  was updated at time  $x$ . We can see from Figure 8 that a global basis is updated frequently, and it is less frequent to update a global basis at a smaller index. Therefore, if the number of shares  $k$  is small, for example  $k = 1$ , each solver will run with almost no information sharing. To benefit from this sharing effect of CMAP-DeepBKZ, it is necessary to have a large number of shares.

**Approximation factors in projected lattices** In Figure 9, we show the approximate factors in projected lattices for a global basis  $\mathbf{S} = (s_1, \dots, s_d)$ , output by CMAP-DeepBKZ after six hours of execution for the SVP lattice  $L$  of dimension  $d = 118$  with seed 2. Specifically, we plot all  $(i, y_i)$  for  $1 \leq i \leq d$ , where  $y_i = \frac{\|s_i^*\|}{\text{GH}(\pi_i(L))}$  denotes the approximate factor in the projected lattice  $\pi_i(L)$  of dimension  $n = d - i + 1$ . (Recall that  $\text{GH}(\pi_i(L)) \approx \lambda_1(\pi_i(L))$  for large  $n \geq 50$ ; however it does not hold for small  $n$ .) Therefore, we focus on indices  $1 \leq i \leq 80$ . We note from Figure 9 that approximate factors at indices  $1 \leq i \leq 16$

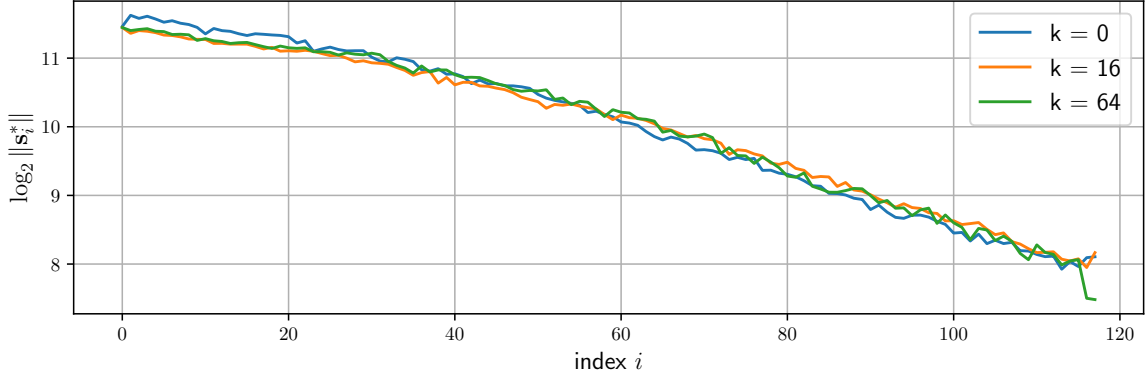


Figure 10: The logarithms of Gram-Schmidt squared norms  $\log_2 \|\mathbf{s}_i^*\|$  of a global basis  $\mathbf{S} = (\mathbf{s}_1, \dots, \mathbf{s}_d)$  output by CMAP-DeepBKZ with the numbers of shares  $k = 0, 16$  and  $64$  after 6 hours execution for an SVP instance in  $d = 118$

are extremely close to 1.0 when the numbers of shares  $k = 16$  and  $64$ . This implies that the first 16 basis vectors of  $\mathbf{S}$  are almost equal to those of an HKZ-reduced basis. (We also note from Figure 9 that  $k = 16$  seems sufficient for dimension  $d = 118$ .)

**GSA shapes** In Figure 10, we show the logarithms of the Gram-Schmidt squared norms  $\log \|\mathbf{s}_i^*\|^2$  of a global basis  $\mathbf{S} = (\mathbf{s}_1, \dots, \mathbf{s}_d)$ , output by CMAP-DeepBKZ with the number of shares  $k$  after six hours execution for the SVP instance in dimension  $d = 118$  with seed 2. We can observe the “head concavity” as pointed out in [9] in both cases with and without sharing (cf., see [2, Figure 1] for an image of the GSA shape by the BKZ reduction algorithm.) Specifically, the log-norms  $\log \|\mathbf{s}_i^*\|^2$  at the first 20 indices for the two cases  $k = 16$  and  $64$  are more concave than for the case  $k = 0$ .

**Remark 3 (Performance difference due to the number of shares)** *Through exhaustive experimentation considering different numbers of shares  $k$  for 95, 100, and 105-dimensional SVPs in Subsection 5.2.1, the results showed little difference in the root Hermite factor when the number of shares  $k > 0$ . In contrast, the value of the enumeration cost and the GSA slope tended to improve as the number of shares  $k$  increased, but converged to similar values for  $k \geq 16$ . The same tendency was observed in the experiment using CMAP-DeepBKZ in 118-dimensional SVPs in Subsection 5.2.2. In addition, as shown in Figure 15, when the number of shares  $k = 64$ , lattice bases update frequently in each solver, and the number of substantial shares is up to 16. This result explains why there are no significant differences between  $k = 16$  and  $64$ . In the following subsections, we mainly use  $k = 16$  in terms of both the output quality and the diversity of CMAP-DeepBKZ.*

### 5.3 Scalability of the number of processes

In this subsection, we show the scalability of CMAP-DeepBKZ in large-scale computing environments. Specifically, we used different computing platforms with a maximum of

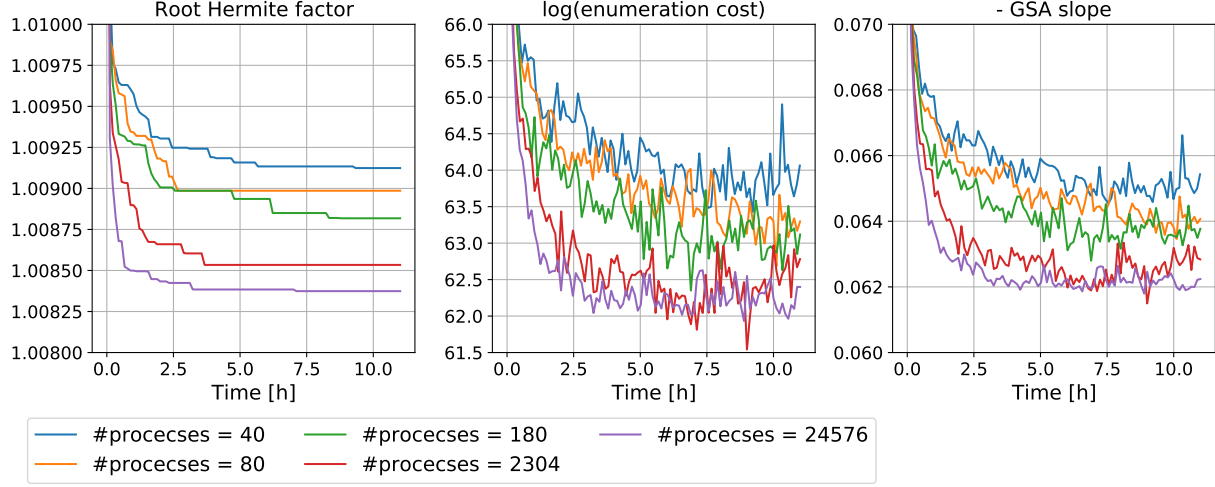


Figure 11: Same as Figure 7, but the dimension is  $d = 120$  and lines in each metric represent difference by different numbers of processes (We used  $k = 16$  as the number of shares)

$p = 24,576$  cores (see Table 3 for details of computing platforms). We ran CMAP-DeepBKZ for 11 hours for every instance of the Darmstadt SVP challenge [28] in two dimensions,  $d = 120$  and  $d = 124$ , with seeds 0–4. More specifically, each solver used an initial blocksize  $\beta = 30$  for DeepBKZ, increasing  $\beta$  by increments of five with the early termination strategy of [11]. (The strategy is also implemented in `fpIII` [36] as an auto-abort option for BKZ.) When a solver reached  $\beta = 50$ , the reduction process was terminated and the solver received a new task (that is, a new basis) from the supervisor to run DeepBKZ again from the beginning. We set  $k = 16$  as the number of short basis vectors shared among solvers, which is a low value that on average exhibited good performance in the experiments of the previous section. In Tables 3 and 4, we show experimental results on the scalability of CMAP-DeepBKZ in the dimensions  $d = 120$  and  $d = 124$ , respectively. We assigned one core to the supervisor except for Emmy and used  $p - 1$  solvers for basis reduction. When using  $p = 24,576$  cores for Emmy, we assigned one node to the supervisor with a sufficient amount of memory, and used  $p - 96 = 24,480$  solvers for basis reduction. In Figure 11, we also show the same as Figure 7, but the dimension is  $d = 120$  and different lines in each metric correspond to different numbers of cores. Because the computing platforms are different, the comparison is not exact; however as shown in Tables 3, 4 and Figure 11, the quality of a global basis improves in every metric as the number of cores is increased. In particular, Table 3 shows that an extremely short lattice vector with an approximate factor close to 1.0 in dimension  $d = 120$  can be found within 11 hours when using  $p = 24,576$  cores for CMAP-DeepBKZ. To evaluate the scalability, we recall from the Gaussian Heuristic that there are roughly  $\alpha^d$  lattice vectors of norms less than  $\alpha \text{GH}(L)$  in a  $d$ -dimensional lattice  $L$  for a constant  $\alpha \geq 1$ . When we evaluate the hardness of an approximate SVP by the number of solutions, the approximate factor  $\alpha = 1.0013$  achieved

Table 3: Results of CMAP-DeepBKZ after 11 hours execution on platforms with the number of processes  $p$  for SVP instances in dimension  $d = 120$  (We used  $k = 16$  as the number of shares, and let  $\mathbf{b}_1$  denote a shortest basis vector of all solver’s bases)

SVP instance	Number of cores	Updated time [h]	Norm of $\mathbf{b}_1$	Approx. factor $\frac{\ \mathbf{b}_1\ }{\text{GH}(L)}$	Root Hermite factor $\gamma^{1/d}$	Machine (Table 1)
seed0	$p = 180$	6.2015	2848.69	1.0288	1.00860	CAL C
seed1		8.2300	2963.32	1.0669	1.00891	
seed2		8.6904	2996.73	1.0785	1.00900	
seed3		4.6942	2898.89	1.0424	1.00871	
seed4		1.6277	2947.42	1.0618	1.00887	
Average				1.0557	1.00882	
seed0	$p = 2,304$	1.1908	2804.94	1.0130	1.00847	ITO
seed1		2.8657	2844.46	1.0241	1.00856	
seed2		1.5187	2896.61	1.0424	1.00871	
seed3		1.4221	2897.27	1.0419	1.00871	
seed4		3.6159	2729.25	0.9833	1.00822	
Average				1.0209	1.00853	
seed0	$p = 24,576$	1.5810	2756.06	0.9954	1.00833	Emmy
seed1		7.0333	2792.47	1.0054	1.00841	
seed2		3.1890	2778.82	1.0001	1.00836	
seed3		0.6497	2842.70	1.0222	1.00855	
seed4		0.6117	2729.25	0.9833	1.00822	
Average				1.0013	1.00837	

by using  $p = 24,576$  processes is  $(1.0557/1.0013)^{120} \approx 572$  times harder than  $\alpha = 1.0557$ , which was attained by  $p = 180$  in dimension  $d = 120$  as shown in Table 3.

In Figure 12 (resp., Figure 13), similar to Figure 7 (resp., Figure 10), we show approximate factors in projected lattices (resp., the logarithms of Gram-Schmidt squared norms) of a global basis in  $d = 120$  according to the different numbers of processes. Because we shared the first 16 basis vectors among the solvers, the plots at the first 16 indices in Figure 12 become closer to 1.0 by increasing the number of processes. Similarly, we see from Figure 13 that the logarithms of the Gram-Schmidt squared norms of a global basis in the first 16 indices are reduced as the number of cores is increased.

## 5.4 Transition of diversity on large-scale execution

We measured the diversity of a set of bases of the solver during large-scale execution with Div defined in Section 4.2. Figure 15 is created from five results of 118-dimensional instances in Section 5.2.2, with six hours executions using 2,304 cores and 16 shared short vectors. Figure 15 shows the three results with different numbers of shared vectors. The left figure shows the transition of the number of overlapping basis vectors, excluding positive and negative differences. Because the solver obtained the global basis from the supervisor at

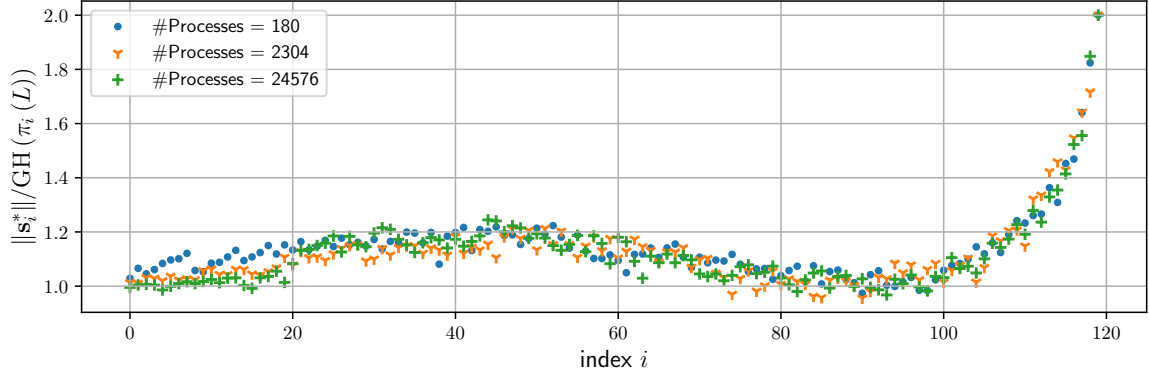


Figure 12: Same as Figure 7, but the dimension is  $d = 120$  and plots represent difference by different numbers of cores (We used  $k = 16$  as the number of shares)

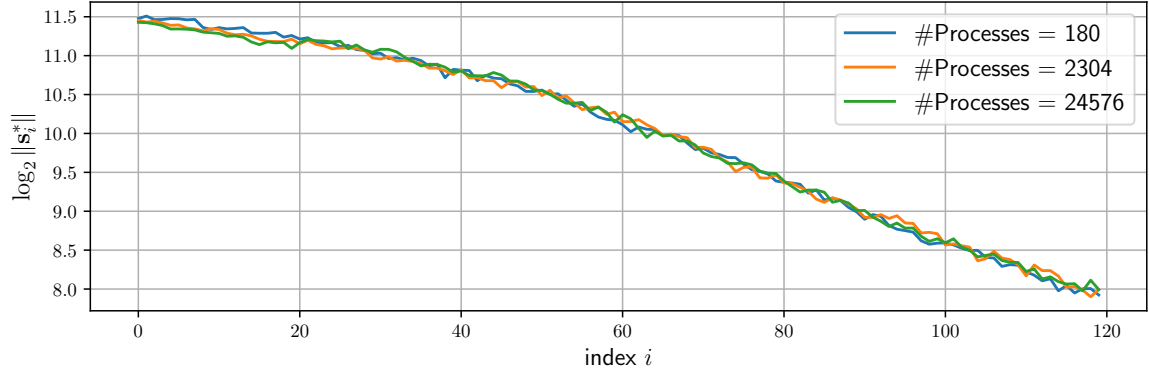


Figure 13: Same as Figure 10, but the dimension is  $d = 120$  and three lines represent different GSA shapes by different numbers of processes (We used  $k = 16$  as the number of shares)

relatively large intervals of 120 seconds, the situation where the top-16 vectors are aligned did not occur during the early calculation time, and the number of overlaps approaches 16 after one hour. The right figure shows the transition of the Div values using the chordal metric. The diversity Div is defined as the average of the diversities for all pairs in the basis set. However, because the size of the basis set is 2,303 for these executions, which is equal to the number of solvers, calculating the diversity for all pairs in this set requires high computation time and is impractical. Therefore, we sampled 100 basis pairs from the basis set and approximated Div by taking the average value of those pairs. This computation of Div was performed every 10 minutes, and it was shown that Div grows smaller as the execution progresses, that is, the diversity of the basis set tends to decrease. However, the transition of Div did not continue to decrease and eventually plateaus, even though the actual number of basis vectors received from the supervisor was larger than 16. This tendency for diversity to plateau was also confirmed in a large-scale experiment using the

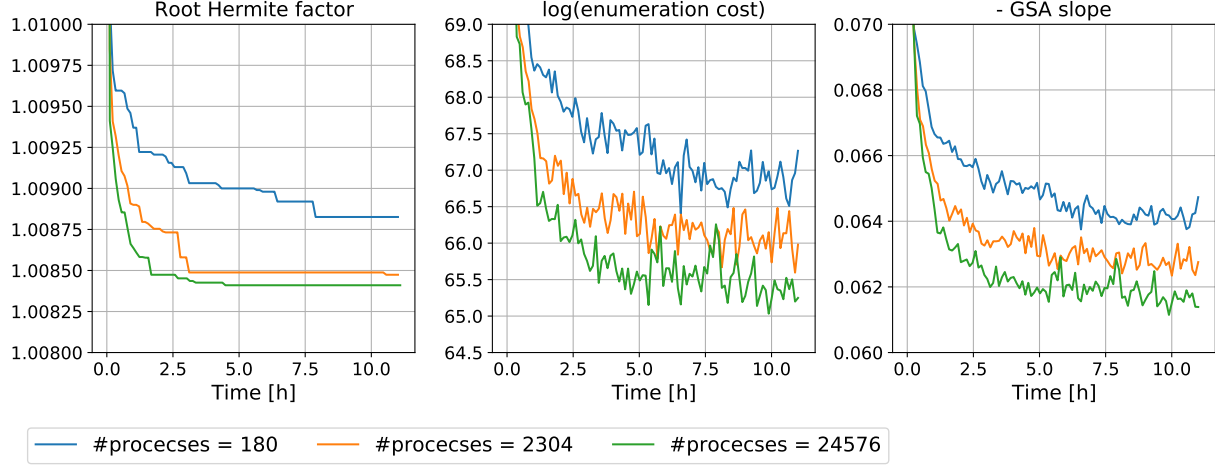


Figure 14: Same as Figure 7, but the dimension is  $d = 124$  and lines in each metric represent difference by different numbers of processes (We used  $k = 16$  as the number of shares)

24,576 cores. Figure 16 was created from the results of experiments utilizing up to 24,576 cores in 11 hours executions on a 120-dimensional SVP in Section 5.3. The figures are the same as Figure 15 but show the diversity transition for the different number of cores. The tendency for diversity to plateau suggests that the diversity of the basis is preserved even in large-scale execution owing to the one-time randomization performed before the lattice basis reduction. Therefore, even in the large-scale computing platform where massive solvers execute the lattice basis reduction in parallel, the computations of the subroutines of the lattice basis reduction hardly overlapped. This result indicates that efficient use of computational resources was achieved in our software.

## 5.5 Massive parallelization experiments with checkpoints and restarts

For CMAP-DeepBKZ, we conducted large-scale experiments on the supercomputer systems Emmy and Lisa (Table 1) with multiple checkpoints and restarts for instances of the Darmstadt SVP challenges [28] in dimensions  $d = 128, 130$  and  $132$ . In Figure 17, we show the transition of the approximation factor of a shortest basis vector in all bases of solver during the execution of CMAP-DeepBKZ. We started with the numbers of shared vectors  $k = 16$  and manually increased  $k$  to 32 when the global basis was no longer being significantly updated. In Table 5, we summarize the final output results of Figure 17. In particular, we succeeded in finding a new solution for the SVP challenge in the dimension  $d = 128$  using an instance with seed 1. It took approximately 57.5 hours to find the new solution, whose norm (resp., approximation factor) is 2812.0 (resp., 0.98470) from Table 5. In contrast, it was reported on the webpage of [28] that it took approximately five months on an iMac core-i7 to find the previous record in the case of  $d = 128$ , the norm (resp.,

Table 4: Same as Figure 3, but the dimension is  $d = 124$ 

SVP instance	Number of processes	Updated time [h]	Norm of $\mathbf{b}_1$	Approx. factor $\frac{\ \mathbf{b}_1\ }{\text{GH}(L)}$	Root Hermite factor $\gamma^{1/d}$	Machine (Table 1)
seed0	$p = 180$	5.8853	3086.00	1.0930	1.00894	CAL C
seed1		4.1859	3082.17	1.0948	1.00896	
seed2		7.8734	2879.06	1.0207	1.00839	
seed3		4.3137	3101.22	1.0996	1.00899	
seed4		2.9052	3045.08	1.0807	1.00885	
Average				1.0778	1.00883	
seed0	$p = 2,304$	2.1351	2978.44	1.0549	1.00866	ITO
seed1		10.489	3015.78	1.0712	1.00878	
seed2		3.0634	2885.80	1.0231	1.00841	
seed3		2.7563	2742.98	0.9726	1.00800	
seed4		1.3161	2921.65	1.0369	1.00852	
Average				1.0317	1.00847	
seed0	$p = 24,576$	3.3615	2892.64	1.0245	1.00842	Emmy
seed1		1.6687	2920.47	1.0374	1.00852	
seed2		3.1216	2854.12	1.0118	1.00832	
seed3		0.7056	2886.65	1.0236	1.00841	
seed4		4.3993	2873.73	1.0199	1.00838	
Average				1.0234	1.00841	

approximation factor) of which was about 2882 (resp., 1.00477). However, the norms of Table 5 in the other dimensions  $d = 130$  and  $132$  do not surpass the current records yet.

**Execution details on Lisa** We describe execution details on Lisa when using 103,680 cores, which is the maximum number of cores used across all computers (Table 1 for computing platforms). We used Lisa for solving SVP instances in dimension 130 with seeds 3 and 7. In both executions, solutions were updated after more than 28 hours of execution (see Table 17). In Figure 18 and 19, we show snapshots of a global basis  $\mathbf{S} = (\mathbf{s}_1, \dots, \mathbf{s}_d)$  in dimension 130 with seed 7 execution. Over the course of the execution, the values of the approximation factor for each  $i$ -th projected lattice  $\|\mathbf{s}_i^*\|/\text{GH}(\pi_i(L))$  grew smaller for indices  $i$  under 32, and approached 1.0 at 100 hours. This implies that a basis close to the HKZ-reduced basis was obtained for the first indexes of the basis. This strict reduction is also clearly shown for GSA shapes in Figure 19. We can see the step difference at the index with exactly  $i = 32$ , which corresponds to the final number of shares  $k$ . While the GSA slope  $\rho$  of the entire basis is  $-0.05867$ , but the  $\rho$  of the sub-basis consisting of  $(\mathbf{s}_1, \dots, \mathbf{s}_{32})$  is  $-0.03685$ , indicating that the first indexes of the basis were more reduced.

**Communication performance** Here, we describe the memory usage and CPU utilization on Lisa supercomputer using  $p = 103,680$  cores for a 130-dimensional instance with seed 7 instance. One node was allocated to a supervisor process, leaving  $p - 96 = 103,524$

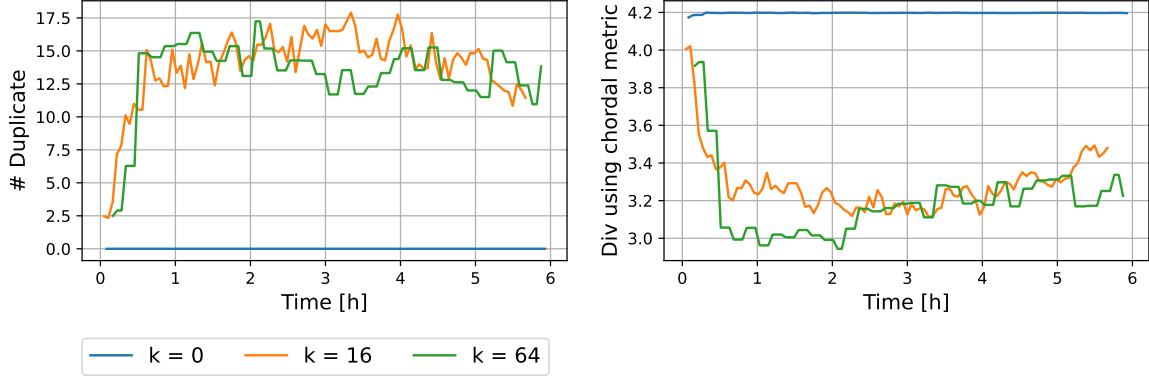


Figure 15: Transition of the diversity of 118-dimensional lattice basis with different the number of shared vectors; left figure is the transition of the number of overlap of basis vectors, right figure is the transition of the Div with Projection metric

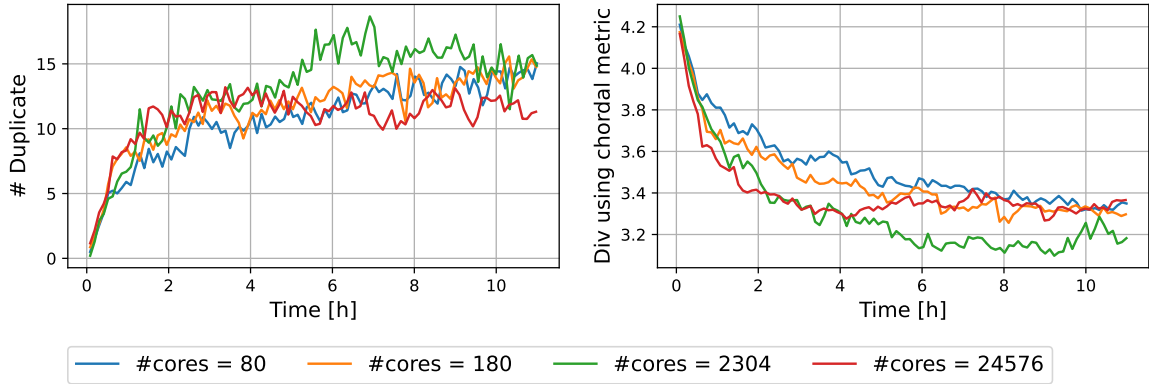


Figure 16: Same as Figure 15, but dimension is 120 and with different the number of cores.

solvers to be created in the remaining nodes. The maximum memory usage in the supervisor (resp., the solver) process was 61.7172 GiB (resp., 0.2274 GiB). Both transitions of the memory usage during the runtime eventually plateaued, aligning with our expectations. Because the amount of memory usage of DeepBKZ in Algorithm 1 does not change, we can maintain low memory usage in the solver process. This implies that the solver process can execute even in a low-memory computational environment. By contract, because the supervisor has the lattice basis information of all solvers in the solver pool, it requires a sufficient amount of memory.

Next, we describe the CPU utilization of the supervisor and solver processes. The ratio of idle time to the total execution time of the solver is 0.9059%, including the communication latency for receiving tasks and lattice vectors from the supervisor process. The ratio of idle time was extremely low, suggesting that the solver process has a high CPU utilization. In the case of the supervisor, the ratio of idle time was 81.36%. This idle time corresponded to the time spent by the supervisor when waiting for a message from

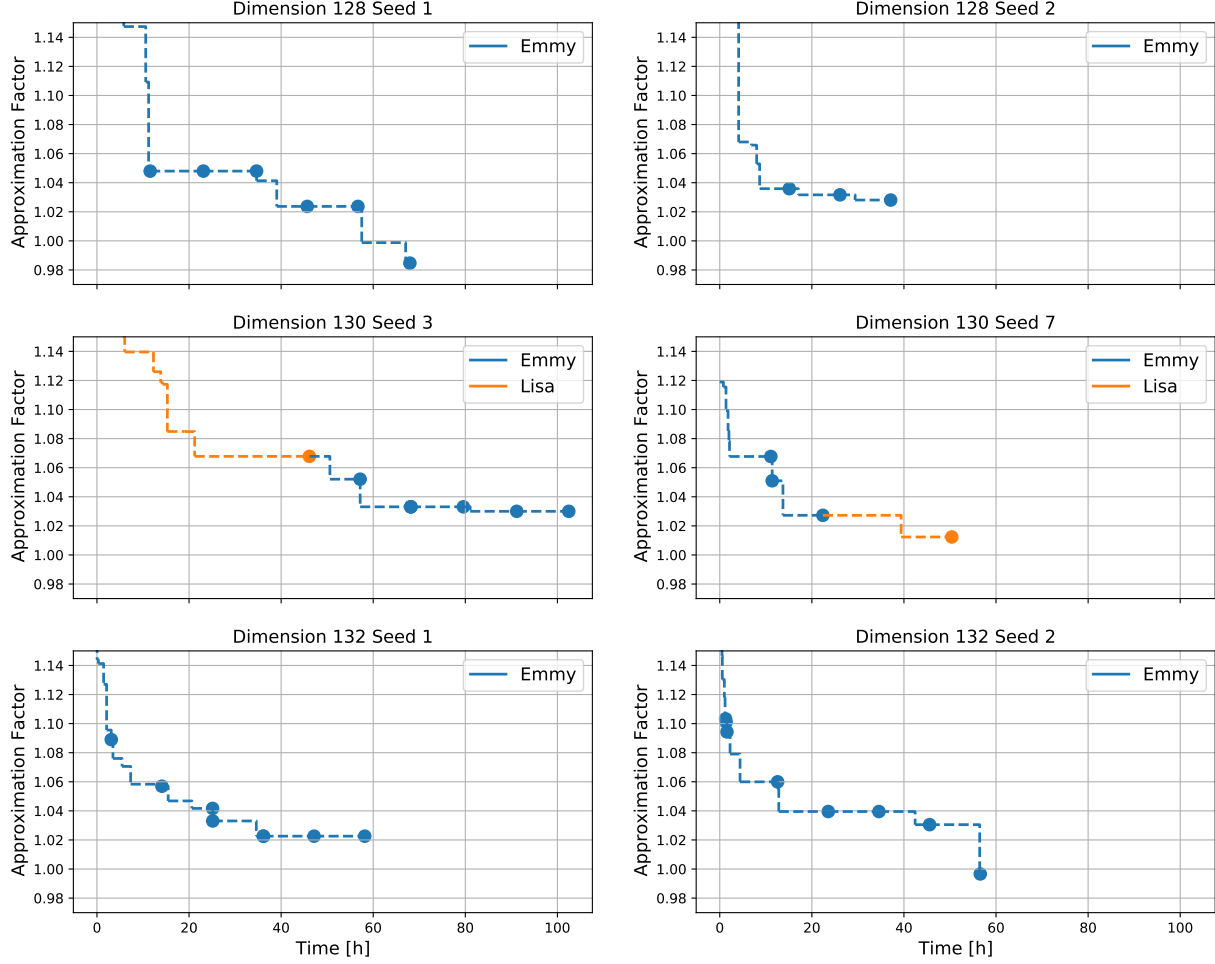


Figure 17: Transition of the approximation factor  $\frac{\|\mathbf{b}_1\|}{\text{GH}(L)}$  of a shortest basis vector  $\mathbf{b}_1$  for SVP instances in dimensions  $d = 128, 130$  and  $132$  (Each dot show the timing of checkpoint-and-restart, and see also Table 5 for a summary)

the solver, and a large idle rate is desirable because it allows the supervisor to process messages from the solver without delay.

Next, to evaluate the stability of our software, we note the checkpointing times of these executions. Specifically, the checkpoint creation times increase along with the number of solvers because our software writes all task data to checkpoint files, including information of all bases of the solvers. While the supervisor is copying the tasks, its message handling is blocked. Therefore, if there is a significant delay when copying, MPI can run out of memory buffers, causing an error. In an execution on Lisa by using 103,584 solvers, it took an average of 1.93 seconds for the supervisor to copy the tasks, and 468.01 seconds for the checkpointing thread created in the supervisor to write the file. We can see that the blocking duration of the supervisor handling was kept extremely short, suggesting that execution by more solvers is possible.

Table 5: Large-scale experimental results of **CMA-DeepBKZ** for SVP instances in dimensions  $d = 128, 130$  and  $132$  ( $\mathbf{b}_1$  denotes a shortest basis vector of all solver’s bases, and “Updated time” is wall time to update final shortest vectors found)

SVP Instance		# of cores*	Updated time [h]	Norm of $\mathbf{b}_1$	Approx. factor $\frac{\ \mathbf{b}_1\ }{\text{GH}(L)}$	Root Hermite factor $\gamma^{1/d}$	Machine*
Dim.	Seed						(Table 1)
128	1 <sup>†</sup>	24,576	57.5	2812.00	0.98470	1.00796	Emmy
	2	24,576	37.1	2947.45	1.02808	1.00830	Emmy
130	3	103,680	81.1	2968.73	1.03001	1.00825	Lisa
	7	103,680	39.4	2914.22	1.01236	1.00811	Lisa
132	1	24,576	34.6	2968.05	1.02260	1.00812	Emmy
	2	24,576	56.5	2899.90	0.99662	1.00818	Emmy

<sup>†</sup> a new solution for the Darmstadt SVP challenge [28] in dimension 128 (see also Table 6 for other dimensions). \* We list the maximum number of cores and machines used for executions, including restarts, and the wall time for the updated time.

Table 6: New solutions for the Darmstadt SVP challenge [28], found by parallel sharing DeepBKZ with the number of shares  $k = 16$  (cf., [33, Table II] for solutions by using  $k = 1$ )

SVP Instance		# of cores	Updated time	Norm of $\mathbf{b}_1$	Approx. factor $\frac{\ \mathbf{b}_1\ }{\text{GH}(L)}$	Root Hermite factor $\gamma^{1/d}$	Machine
Dim.	Seed						(Table 1)
103	3	144	52.3 m	2581.65	0.97168	1.00875	CAL A
109	2	144	49.8 m	2559.17	0.96465	1.00845	CAL A
113	5	144	1.21 h	2621.54	0.97459	1.00840	CAL A
124	2	24,576	2.85 h	2826.79	1.00215	1.00824	Emmy
128	1	24,576	57.5 h	2812.00	0.98470	1.00796	Emmy

**New solutions for the Darmstadt SVP challenge** In Table 6, we list new solutions proposed by **CMA-DeepBKZ** for the Darmstadt SVP challenge [28]. For each dimension  $d = 103, 105, 107, 109, 113$  and  $114$ , we performed **CMA-DeepBKZ** with the number of shares  $k = 16$  for 10 instances from seeds 0 to 9, and succeeded in finding new solutions for dimensions 103, 109 and 113. For dimension 124 (resp., 128), we found a new solution by executing five instances with seeds ranging from zero to four (resp., two instances of seeds 1 and 2) on Emmy. In [33], new solutions were found by parallel DeepBKZ with  $k = 1$  for SVP instances in dimensions up to 127. For dimension  $d = 127$ , it took approximately 147 hours of execution on several supercomputer systems with up to 91,200 cores (see [33, Tables II and III]). In contrast, Table 6 shows that it took about 57.5 hours for  $d = 128$  by **CMA-DeepBKZ** with  $k = 16$  on Emmy with 24,576 processes. Such comparisons provide experimental evidence supporting the efficacy of sharing short basis vectors in parallel DeepBKZ.

**Comparison with G6K** We provide a comparison with **G6K** [3], the state-of-the-art SVP solver using advanced sieve algorithms as described in Subsection 1.3. **G6K** adopts

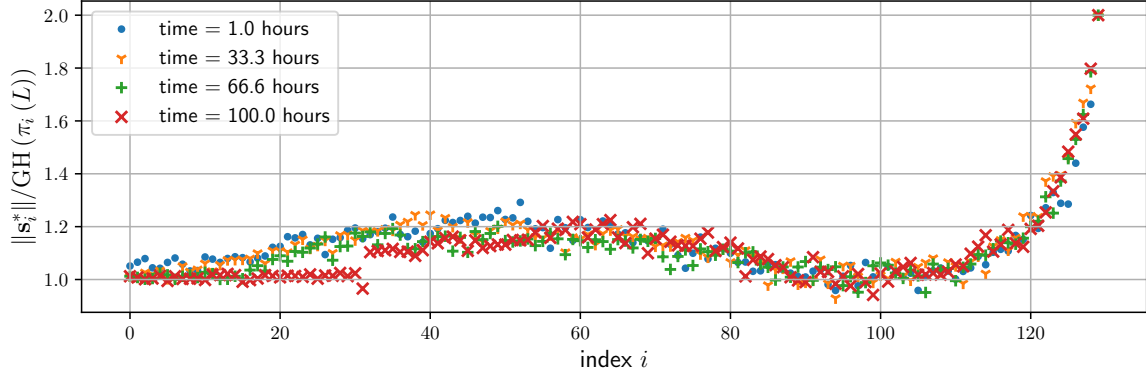


Figure 18: Plots of approximation factors in projected lattices  $\|\mathbf{s}_i^*\|/\text{GH}(\pi_i(L))$  for a global basis  $\mathbf{S} = (\mathbf{s}_1, \dots, \mathbf{s}_d)$  output by CMAP-DeepBKZ of a lattice  $L$  of dimension  $d = 130$  with seed = 7 of SVP challenge instance after 1.0, 33.3, 66.6, 100 hours executions, and the final numbers of shares  $k = 32$

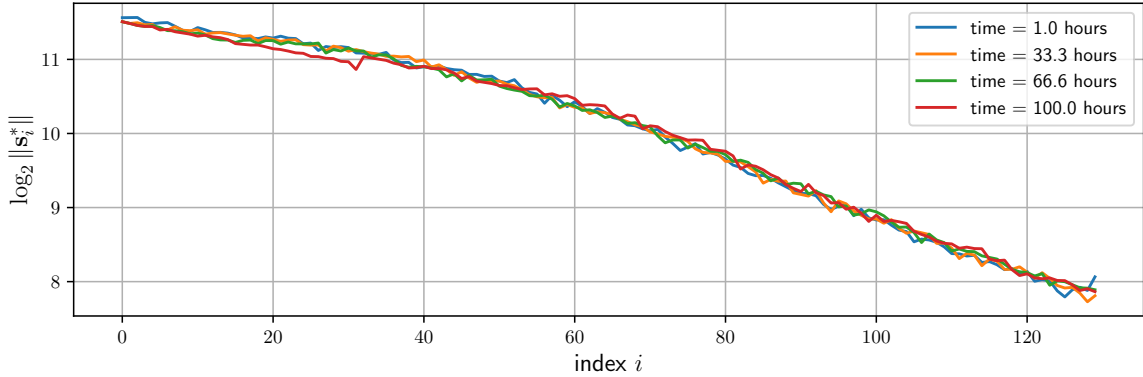


Figure 19: The logarithms of Gram-Schmidt squared norms  $\log_2 \|\mathbf{s}_i^*\|$  of a global basis  $\mathbf{S} = (\mathbf{s}_1, \dots, \mathbf{s}_d)$  of a lattice  $L$  same as Figure 18.

the sub-sieve strategy of [14]. For a  $d$ -dimensional lattice  $L$ , it runs a sieve algorithm in a projected lattice  $\pi_k(L)$  of dimension  $m = d - k + 1$  to find a significantly large number of short *projected* lattice vectors, and lifts them into vectors in the whole lattice  $L$ . Such lattice vectors do not always include shortest vectors in  $L$ ; however, some of them can be short enough to have approximation factors within 1.05 for entering the hall of fame of the SVP challenge. It was reported in [3, Table 2] that it took about 11.6 (resp., 11.8 and 14.7) days to find a solution of the SVP challenge in dimension  $d = 151$  (resp., 153 and 155) by using the maximum sieving dimension  $m = 123$  (resp., 124 and 127). According to the latest result [15, Table 1] for a GPU implementation of sieve algorithms inside G6K, it took about 51.6 days on a server with four NVIDIA Turing GPUs with 1.5TB of RAM for an SVP instance in  $d = 180$  by using  $m = 150$ . Note that the current SVP records in  $d \geq 150$  have approximation factors around 1.03 or 1.04, they must not be the shortest.

Because we do not use the sub-sieve strategy, it is reasonable to compare CMAP-DeepBKZ in dimension  $d$  with G6K in the maximum sieving dimension  $m$ . As shown in Tables 5 and 6, the performance of CMAP-DeepBKZ in dimensions around  $d = 130$  is faster than that of G6K around  $m = 130$  in [3, Table 2] if we ignore the difference of computing resources. In contrast, the performance in [15, Table 1] is faster than CMAP-DeepBKZ due to a GPU-implementation for sieve algorithms. However, sieve algorithms require exponential-space in  $m$ . Indeed, it is reported in [15] that about 1.4TB of RAM was required for finding an SVP solution in  $d = 180$  using  $m = 150$ . On the other hand, CMAP-DeepBKZ adopts enumeration for SVP oracles in blocksize  $\beta$ , and its space-complexity is polynomial with respect to  $\beta$ . In particular, CMAP-DeepBKZ has sufficient performance even with small blocksize such as  $\beta = 30$ . This implies that CMAP-DeepBKZ can be practically applied to large-scale computers with minimal memory footprint and no memory limitation.

## 6 Conclusion

We developed a software using the CMAP-LAP framework [34] for massively parallel execution of a BKZ-type reduction algorithm. Our software enables us to simultaneously execute a reduction algorithm on randomized bases by sharing short basis vectors among solvers to accelerate the reduction process in every solver. We also evaluated the diversity of reduced bases using Grassmann metrics and verified that the randomness of bases is highly unlikely to be lost during the execution of parallel reduction when sharing  $k \leq 64$  short basis vectors for high-dimensional lattices (Figures 2, 3 and 15). Furthermore, we demonstrated through our experiments that sharing  $k = 16$  short basis vectors is effective in both the output quality and the performance of CMAP-DeepBKZ using our software with DeepBKZ [37] as a reduction algorithm. Our experiments (Table 5) showed that CMAP-DeepBKZ with small blocksize around  $\beta = 30$ –40 can find a very short vector close to the shortest in a lattice of dimension  $d = 132$  within 100 hours on supercomputers using up to 103,680 cores, without using other strategies such as the sub-sieve of [14] adopted by G6K [3]. Specifically, it took approximately 57.5 hours using 24,576 cores to find a new solution of the Darmstadt SVP challenge in dimension  $d = 128$  (Table 6).

## Acknowledgements

This research project was supported by the Japan Science and Technology Agency (JST), the Core Research of Evolutionary Science and Technology (CREST), the Center of Innovation Science and Technology based Radical Innovation and Entrepreneurship Program (COI Program), JSPS KAKENHI Grant Number JP21H04599, JP20H04142, Japan, the German Research Foundation (DFG) through the project HPO-Navi (fund number 391087700): Sustainable Infrastructures for Archiving and Publishing High-Performance Optimization Software, the Research Campus MODAL funded by the German Federal Ministry of Education and Research (fund number 05M20ZBM). This work was also sup-

ported the National High Performance Computing Center at the Zuse Institute Berlin (NHR@ZIB). We are grateful to the supercomputer staff, especially Matthias Lauter and Tobias Watermann.

## References

- [1] Ajtai, M.: Generating hard instances of lattice problems. In: Symposium on Theory of Computing (STOC 1996), pp. 99–108. ACM (1996)
- [2] Albrecht, M., Ducas, L.: Lattice attacks on NTRU and LWE: A history of refinements. Cryptology ePrint Archive: Report 2021/799 (2021)
- [3] Albrecht, M., Ducas, L., Herold, G., Kirshanova, E., Postlethwaite, E.W., Stevens, M.: The general sieve kernel and new records in lattice reduction. In: Advances in Cryptology–EUROCRYPT 2019, *Lecture Notes in Computer Science*, vol. 11477, pp. 717–746. Springer (2019)
- [4] Albrecht, M.R., Curtis, B.R., Deo, A., Davidson, A., Player, R., Postlethwaite, E.W., Virdia, F., Wunderer, T.: Estimate all the {LWE, NTRU} schemes! In: Security and Cryptography for Networks (SCN 2018), *Lecture Notes in Computer Science*, vol. 11035, pp. 351–367 (2018)
- [5] Barg, A., Nogin, D.Y.: Bounds on packings of spheres in the grassmann manifold. *IEEE Transactions on Information Theory* **48**(9), 2450–2454 (2002)
- [6] Björck, Å., Golub, G.H.: Numerical methods for computing angles between linear subspaces. *Mathematics of computation* **27**(123), 579–594 (1973)
- [7] Bremner, M.R.: Lattice basis reduction: An introduction to the LLL algorithm and its applications. CRC Press (2011)
- [8] Burger, M., Bischof, C., Krämer, J.: p3Enum: A new parameterizable and shared-memory parallelized shortest vector problem solver. In: Computational Science–ICCS 2019, *Lecture Notes in Computer Science*, vol. 11540, pp. 535–542. Springer (2019)
- [9] Chen, H.: A measure version of Gaussian heuristic. IACR Cryptology ePrint Archive: Report 2016/439 (2016)
- [10] Chen, Y.: Réduction de réseau et sécurité concrete du chiffrement complètement homomorphe. Ph.D. thesis, Paris 7 (2013)
- [11] Chen, Y., Nguyen, P.Q.: BKZ 2.0: Better lattice security estimates. In: Advances in Cryptology–ASIACRYPT 2011, *Lecture Notes in Computer Science*, vol. 7073, pp. 1–20. Springer (2011)

- [12] Dagdelen, Ö., Schneider, M.: Parallel enumeration of shortest lattice vectors. In: Euro-Par 2010–Parallel Processing, *Lecture Notes in Computer Science*, vol. 6272, pp. 211–222. Springer (2010)
- [13] Deutsch, P., Gailly, J.L.: Zlib compressed data format specification version 3.3. Tech. rep., RFC 1950, May (1996)
- [14] Ducas, L.: Shortest vector from lattice sieving: A few dimensions for free. In: Advances in Cryptology–EUROCRYPT 2018, *Lecture Notes in Computer Science*, vol. 10820, pp. 125–145. Springer (2018)
- [15] Ducas, L., Stevens, M., van Woerden, W.: Advanced lattice sieving on GPUs, with tensor cores. In: Advances in Cryptology–EUROCRYPT 2021, *Lecture Notes in Computer Science*, vol. 12697, pp. 249–279. Springer (2021)
- [16] Edelman, A., Arias, T.A., Smith, S.T.: The geometry of algorithms with orthogonality constraints. *SIAM journal on Matrix Analysis and Applications* **20**(2), 303–353 (1998)
- [17] Gama, N., Nguyen, P.Q.: Predicting lattice reduction. In: Advances in Cryptology–EUROCRYPT 2008, *Lecture Notes in Computer Science*, vol. 4965, pp. 31–51. Springer (2008)
- [18] Gama, N., Nguyen, P.Q., Regev, O.: Lattice enumeration using extreme pruning. In: Advances in Cryptology–EUROCRYPT 2010, *Lecture Notes in Computer Science*, vol. 6110, pp. 257–278. Springer (2010)
- [19] Golub, G.H., Van Loan, C.F.: Matrix Computations, forth edn. The Johns Hopkins University Press (1996)
- [20] Hermans, J., Schneider, M., Buchmann, J., Vercauteren, F., Preneel, B.: Parallel shortest lattice vector enumeration on graphics cards. In: Progress in Cryptology–AFRICACRYPT 2010, *Lecture Notes in Computer Science*, vol. 6055, pp. 52–68. Springer (2010)
- [21] Joux, A.: A tutorial on high performance computing applied to cryptanalysis (invited talk). In: Advances in Cryptology–EUROCRYPT 2012, *Lecture Notes in Computer Science*, vol. 7237, pp. 1–7. Springer (2012)
- [22] Kannan, R.: Minkowski’s convex body theorem and integer programming. *Mathematics of operations research* **12**(3), 415–440 (1987)
- [23] Kuo, P.C., Schneider, M., Dagdelen, Ö., Reichelt, J., Buchmann, J., Cheng, C.M., Yang, B.Y.: Extreme enumeration on GPU and in clouds. In: Cryptographic Hardware and Embedded Systems–CHES 2011, *Lecture Notes in Computer Science*, vol. 6917, pp. 176–191. Springer (2011)

- [24] Lenstra, A.K., Lenstra, H.W., Lovász, L.: Factoring polynomials with rational coefficients. *Mathematische Annalen* **261**(4), 515–534 (1982)
- [25] Nguyen, P.Q.: Hermite’s constant and lattice algorithms. In: *The LLL Algorithm*, pp. 19–69. Springer (2009)
- [26] Pohmann, S., Stevens, M., Zumbrägel, J.: Lattice enumeration on GPUs for fplll. *IACR ePrint* 2021/430 (2021)
- [27] Ralphs, T., Shinano, Y., Berthold, T., Koch, T.: *Parallel Solvers for Mixed Integer Linear Optimization*, pp. 283–336. Springer International Publishing, Cham (2018). DOI 10.1007/978-3-319-63516-3{\\\_}8. URL [https://doi.org/10.1007/978-3-319-63516-3\\_8](https://doi.org/10.1007/978-3-319-63516-3_8)
- [28] Schneider, M., Gama, N., Baumann, P., Nobach, L.: SVP challenge (2010). URL: <http://latticechallenge.org/svp-challenge> (2010)
- [29] Schnorr, C.P.: Block Korkin-Zolotarev bases and successive minima. *International Computer Science Institute* (1992)
- [30] Schnorr, C.P.: Lattice reduction by random sampling and birthday methods. In: *Symposium on Theoretical Aspects of Computer Science (STACS 2003)*, *Lecture Notes in Computer Science*, vol. 2607, pp. 145–156. Springer (2003)
- [31] Schnorr, C.P., Euchner, M.: Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical programming* **66**, 181–199 (1994)
- [32] Shinano, Y.: Ug: Ubiquity generator framework. <http://ug.zib.de/>
- [33] Tateiwa, N., Shinano, Y., Nakamura, S., Yoshida, A., Kaji, S., Yasuda, M., Fujisawa, K.: Massive parallelization for finding shortest lattice vectors based on ubiquity generator framework. In: *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–15. IEEE (2020)
- [34] Tateiwa, N., Shinano, Y., Yamamura, K., Yoshida, A., Kaji, S., Yasuda, M., Fujisawa, K.: CMAP-LAP: Configurable massively parallel solver for lattice problems “in press”. In: *2021 IEEE 28th International Conference on High Performance Computing, Data, and Analytics (HiPC)*
- [35] Teruya, T., Kashiwabara, K., Hanaoka, G.: Fast lattice basis reduction suitable for massive parallelization and its application to the shortest vector problem. In: *Public Key Cryptography (PKC 2018)*, *Lecture Notes in Computer Science*, vol. 10769, pp. 437–460. Springer (2018)
- [36] The FPLLL development team: fplll, a lattice reduction library (2016). URL <https://github.com/fplll/fplll>

- [37] Yamaguchi, J., Yasuda, M.: Explicit formula for Gram-Schmidt vectors in LLL with deep insertions and its applications. In: Number-Theoretic Methods in Cryptology (NuTMiC 2017), *Lecture Notes in Computer Science*, vol. 10737, pp. 142–160. Springer (2017)
- [38] Yasuda, M.: A survey of solving SVP algorithms and recent strategies for solving the SVP challenge. In: International Symposium on Mathematics, Quantum Theory, and Cryptography, pp. 189–207. Springer (2021)
- [39] Yasuda, M., Nakamura, S., Yamaguchi, J.: Analysis of DeepBKZ reduction for finding short lattice vectors. *Designs, Codes and Cryptography* **88**, 2077–2100 (2020)
- [40] Yasuda, M., Yamaguchi, J.: A new polynomial-time variant of LLL with deep insertions for decreasing the squared-sum of Gram-Schmidt lengths. *Designs, Codes and Cryptography* **87**, 2489–2505 (2019)
- [41] Yu, Y., Ducas, L.: Second order statistical behavior of LLL and BKZ. In: Selected Areas in Cryptography (SAC 2017), *Lecture Notes in Computer Science*, vol. 10719, pp. 3–22. Springer (2017)

---

**Algorithm 2** Processing flow of the supervisor

---

```
1: procedure supervisor(B) ▷ B: instance basis
2:   S  $\leftarrow$  B; ▷ Set initial the global basis S
3:   seed  $\leftarrow$  0;
4:   for  $i = 1 \rightarrow m$  do
5:     C  $\leftarrow$  randomize(B, seed); seed  $\leftarrow$  seed + 1;
6:     Send task (C, parameters) to  $i$ -rank solver; ▷ Send initial tasks to solvers
7:     SolverPool[ $i$ ]  $\leftarrow$  (C, parameters);
8:   end for
9:   while iProbe(source, tag) do
10:    if tag is SolverState then
11:      Receive Status (B,  $\beta$ ) from the source-rank solver;
12:      ▷ B is basis and  $\beta$  is blocksize of DeepBKZ
13:      Update task of source-rank solver in the solver pool using (B,  $\beta$ );
14:      if B < S $k$  then
15:        S  $\leftarrow$  B; ▷ Update the global basis
16:      else if B > S $k$  then
17:        Send S $k$  to the source-rank solver;
18:      end if
19:      Send notification to the source-rank solver;
20:    end if
21:    if tag is Termination then
22:      C  $\leftarrow$  randomize(B, seed); seed  $\leftarrow$  seed + 1;
23:      Send task (C, parameters) to the source-rank solver;
24:      SolverPool[source]  $\leftarrow$  (C, parameters);
25:    end if
26:    if current time reaches the checkpoint time then
27:      Serialize SolverPool, compress it, and write it to checkpoint file;
28:      ▷ Create a checkpoint file
29:    end if
30:    if current time reaches the time limit then
31:      break;
32:    end if
33:  end while
34: end procedure
```

---

---

**Algorithm 3** Reduction algorithm in solver

---

```
1: procedure Reduction( $\mathbf{B}, \beta$ )
2:
3:   Set  $t_s$  to next status sending time;
4:   while Reduction has not finished do
5:      $\mathbf{B} \leftarrow \text{subroutine}(\mathbf{B}, \beta)$ ; ▷ Subroutine of reduction algorithm
6:     if current time  $> t_s$  then
7:       Send a status  $(\mathbf{B}, \beta)$  to supervisor with SolverState tag;
8:       Wait a notification from supervisor;
9:       if solver receives the global sub-basis  $\mathbf{S}_k = (\mathbf{s}_1, \dots, \mathbf{s}_k)$  then;
10:        if  $\mathbf{S}_k < \mathbf{B}$  then
11:          for  $j = 1 \rightarrow k$  do
12:             $l \leftarrow$  minimum index  $h$  satisfies  $\|\pi_h(\mathbf{s}_j)\| < \|\mathbf{b}_h^*\|$ ;
13:             $\mathbf{B} \leftarrow \text{LLL}((\mathbf{b}_1, \dots, \mathbf{b}_{l-1}, \mathbf{s}_j, \mathbf{b}_l, \dots, \mathbf{b}_d))$ ;
14:            ▷ Merge the global sub-basis into its own basis
15:          end for
16:        end if
17:      end if
18:      Update  $t_s$  to next status sending time;
19:    end if
20:  end while
21:  Send Termination tag to supervisor;
22: end procedure
```

---