Development and Numerical Experiments of Massively Parallel Framework and Software for Shortest Vector Problem

立岩, 斉明

KYUSHU UNIVERSITY

DOCTORAL THESIS

Development and Numerical Experiments of Massively Parallel Framework and Software for Shortest Vector Problem

---

Author:
Nariaki TATEIWA

Supervisor:
Prof. Katsuki FUJISAWA

A thesis submitted in fulfillment of the requirements
for the degree of Doctor of Mathematics

in the

Graduate School of Mathematics

January 17, 2022

KYUSHU UNIVERSITY

Abstract

Graduate School of Mathematics

Doctor of Mathematics

Development and Numerical Experiments of Massively Parallel Framework and Software for Shortest Vector Problem

by Nariaki TATEIWA

Lattice-based cryptography has received attention as next-generation encryption because it is believed to be secure against attacks by classical and quantum computers. Its essential security depends on the hardness of solving the shortest vector problem (SVP), the primitive lattice problems. In cryptography, to determine security levels, it is becoming significantly essential to estimate the hardness of the SVP by high-performance parallel computing.

Several algorithms have been developed for SVP, however there is no single definite algorithm. They has different computational profiles; some suffer from super-exponential time, and others require exponential space. This motivated us to develop a novel framework for the parallelization of SVP solvers for the clever coordination of different algorithms that run massively in parallel. With our flexible framework, heterogeneous modules run asynchronously parallel on a large-scale distributed system while exchanging information, drastically boosting overall performance. We also implement full checkpoint-and-restart functionality, which is vital to high-dimensional SVP. The parallel scheme in our framework was designed to facilitate the implementation of past and future parallelization methods. Through numerical experiments with up to 103,680 cores, we evaluated the performance and stability of our framework and demonstrated its high capability for future massive-scale experiments.

In addition, by taking full advantage of the features of our framework, we also developed the software for SVP. We have implemented our proposed a new distributed and asynchronous parallel reduction algorithm, DeepBKZ, which is an enhancement of the block Korkine-Zolotarev (BKZ) reduction algorithm. Randomized copies of a basis are distributed to massively cores and reduced independently in parallel, while some basis vectors are shared asynchronously among all processes. There is a trade-off between randomization and information sharing; if too much information was shared, all processes would work on the same problem and the benefit of parallelization would be lost. To monitor the balance between randomness and sharing, we propose a metric to quantify the variety of bases. We demonstrate by experiments the efficacy of our proposed parallel algorithm and our implementation in both performance and scalability.

Keywords: Shortest vector problem, Parallel computing, Lattice basis reduction, Generalized UG framework.

Contents

List of Symbols

$\mathbb{N}, \mathbb{Z}, \mathbb{R}$	sets of natural, integer, real numbers
$\mathbb{Z}^n, \mathbb{R}^n$	vector-space of dimension n
$\mathbf{x}$	row vector
$\|\mathbf{x}\|$	Euclidean norm of vector $\mathbf{x}$
$\mathbf{B}$	matrix (composed from vectors, row-wise)
$\mathbf{b}_i$	i^{th} row of matrix A
$\text{diag}(\mathbf{b}_1, ..., \mathbf{b}_n)$	diagonal matrix B
$\mathbf{0}$	all-zeros vector
$\mathbf{I}_n$	$n \times n$ identity matrix
$\log x$	$\ln x$, the natural logarithm with base e
$\text{Span}(\mathbf{W})$	span of row vectors of $\mathbf{W}$
$\mathbf{W}^\perp$	orthogonal complement of $\text{Span}(\mathbf{W})$
$\|S\|$	cardinality of set S
γ_m	m-dimensional Hermite constant

Chapter 1

Introduction

1.1 Background

A *lattice L* is the set of all integral combinations of linearly independent vectors in the Euclidean space $\mathbb{R}^n$. In the past few years, lattices have attracted considerable interest in cryptography. In particular, with the recent development of quantum computers, since 2015, the US National Institute of Standards and Technology (NIST) started developing new standards for *post-quantum cryptography (PQC)* and called for proposals to prepare information security systems that can resist quantum computers [SN]. (cf., The most popular cryptographic systems, such as RSA, DSA, and ECDSA, could be broken by Shor's algorithms [Sho94] with the use of large-scale quantum computers.) In 2021, NIST allowed 7 finalists and 8 alternates for the third round of the NIST PQC Standardization Process, among which 7 were based on lattices.

Lattice problems are a class of discrete optimization problems whose objective functions are defined on the set of lattice points or the set of lattice bases. The most fundamental instances of the lattice problems are the *Shortest Vector Problem (SVP)* and *Closest Vector Problem (CVP)*. SVP asks to find the shortest non-zero vector in a given lattice, and CVP asks to find the closest vector in a given lattice to a given vector. Lattice problems are believed to be computationally hard with both classical and quantum algorithms [Cai00] and have been used to construct various cryptosystems [Pei16], including PQC. More specifically, the security of many cryptosystems, such as Goldreich-Goldwasser-Halevi (GGH) cryptosystem and NTRU encryption schemes, is based on the hardness of an approximate variant of SVP and CVP. Therefore, it is important for cryptanalysis to know the limits of solving these lattice problems (see [Jou12] for cryptanalysis using high-performance computing). However, in a distributed computing platform, efficient use of a large number of MPI processes requires appropriate control of the behavior of many processes, memory usage, and communication, and this is very costly for implementation, testing and debugging. In fact, there are only a few software fore SVP that can work in distributed computing platform. It is also important to use parallel strategies that take advantage of properties of lattice algorithms, which is suitable for distributed computing platform.

There are three basic families of lattice algorithms that have been developed to solve practical lattice problems: lattice basis reduction, enumeration (ENUM), and sieve. These algorithms have advantages and disadvantages, and there is no single definite algorithm for lattice problems. Sieve and ENUM are the algorithms perform an exhaustive search of all the short lattice vectors, whose number is exponential in the lattice dimension. Sieve algorithm searches for the shortest vector by repeatedly storing short differences between the short lattice vectors. A high-dimensional SVP instance requires numerous vectors to be stocked. Specifically, it requires a memory that is exponential in the dimension of the input lattice. According to [Alb+19, Table 2], G6K, implements a variety of basis reduction and sieve algorithms, uses approximately 246 GB of memory for solving 127-dimensional SVP instances. In contrast, ENUM has space-complexity is polynomial in the lattice dimension, but it is asymptotically slower than the sieve algorithm. Lattice basis reduction aim to convert lattice basis whose vectors are nearly-orthogonal. This process can find short vectors quickly, but it cannot guarantee to output the shortest lattice vectors.

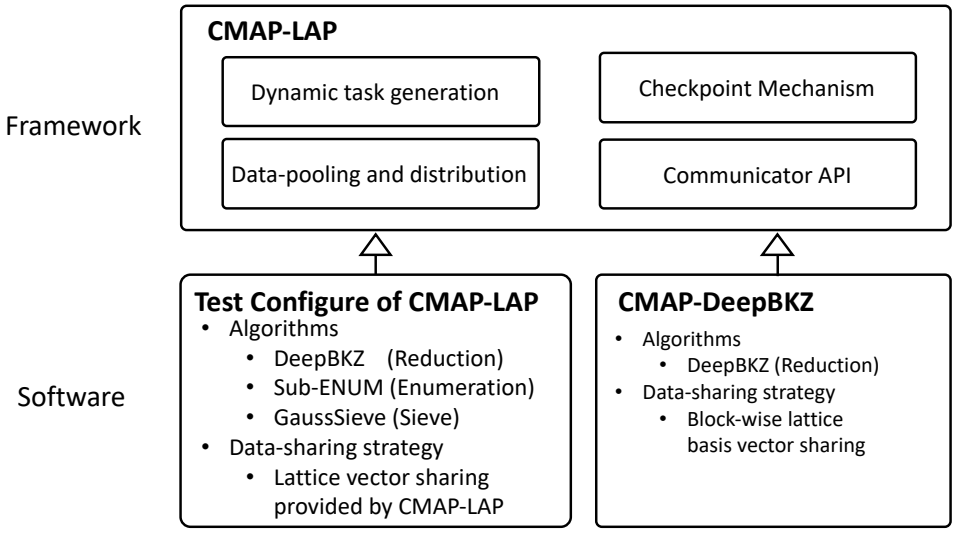

FIGURE 1.1: Relationship of CMAP-LAP and CMAP-DeepBKZ

1.2 Contribution to parallelization of SVP

Existing solvers for lattice problem are limited to a fixed set of algorithms and lack in flexibility. There are two main obstacles in developing a large-scale multi-paradigm solver for suitable in distributed computing platform: the need for an efficient high-level information-sharing scheme across different algorithms, and an adaptive task selection and distribution strategy for hundreds of thousands of processes.

The main contribution of this thesis is to provide solutions to overcome these obstacles and develop a flexible framework to make various algorithms work cooperatively on a large-scale distributed computing platform. By exploiting the mathematical properties of lattice, a clever vector pooling scheme is introduced to minimize the amount of information communicated among processes. To implement our parallel strategy, we used *Generalized UG* (UG version 1.0 RC) that is extended the well-recognized *Ubiquity Generator (UG) framework* [Ug] for Branch-and-Bound (B&B) algorithms. Based on *Generalized UG*, we have built a solid backbone to manage hundreds of thousands of processes running heterogeneous algorithms in parallel, where the assignment of algorithms and their parameters can be adaptively tuned according to the available resources and the progress of the whole system. *Configurable Massively Parallel Solver for Lattice Problems (CMAP-LAP)* is the framework for massively parallel strategies for lattice problems. It is designed to facilitate the implementation of new parallel strategy ideas based on this framework. We have built a solid backbone to manage hundreds of thousands of processes running heterogeneous algorithms in parallel, where the assignment of algorithms and their parameters can be adaptively tuned according to the available resources and the progress of the whole system.

Using CMAP-LAP framework, we also developed the new parallel software *Configurable Massively Parallel Solver for DeepBKZ (CMAP-DeepBKZ)* specialized for the lattice reduction algorithm, this is also the main our contribution. CMAP-DeepBKZ uses a new parallel strategy of DeepBKZ, a variant of lattice basis reduction, to share a part of the lattice basis by taking advantage of parallel computing and the information sharing feature of CMAP-LAP. We have analyzed the performance of this software in detail by experiments on a large number of cores.

We show the relationship between CMAP-LAP framework and CMAP-DeepBKZ software in Figure 1.1.

1.2.1 CMAP-LAP: Framework for lattice problems with massively parallelization

CMAP-LAP [Tat+21] is a generic framework of parallelization for lattice algorithms, including SVP. It covers parallelization of reduction, enumeration and sieve algorithms, and can

run them cooperatively on a large-scale computational platform by supervisor-worker parallel style. Given an input instance, a supervisor distributes randomized instances to all solvers, and all worker processes can execute multiple kinds of solvers and multi-parallel solvers in a heterogeneous. In addition, the supervisor stores lattice bases and vectors in data containers. Using those data containers, each solver can send and receive a lattice basis and vectors asynchronously with small communication overhead.

The features of CMAP-LAP are summarized as follows:

- We propose a novel parallel and multi-algorithm scheme for lattice problems, in which several different single- or multi-rank solvers work cooperatively while sharing information efficiently with other solvers even on a large-scale computing platform. To realize the scheme, CMAP-LAP is developed entirely from scratch by fully utilizing the features of the Generalized UG.
- The testing software using CMAP-LAP with 103,680 cores stably and continuously ran for more than 42 hours. We tested features of CMAP-LAP in several environments with different scales and configurations.
- Each process asynchronously performs various lattice algorithms in coordination while sharing information. Processes for different algorithms are adaptively allocated, and their parameters are tuned according to the available resources, current progress, and estimated time for finding a solution. In particular, our accurate estimation of memory usage has drastically improved the stability and scalability.
- The high-level checkpoint-and-restart functionality is implemented to make it possible to save and resume even on different architectures and platforms of various sizes.
- The efficient information-sharing scheme is developed based on the properties of lattice problems, and is backed with blocking and non-blocking communication mechanisms.
- Highly modular architecture allows one to incorporate new algorithms easily into the system. Existing implementations that work only in a shared-memory environment can work as modules of CMAP-LAP, which run massively in parallel.

1.2.2 CMAP-DeepBKZ: Software for DeepBKZ with massively parallelization

We developed software specialized for massive parallelization of lattice basis reduction. Specifically, we parallelize DeepBKZ [YY17] 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 CMAP-DeepBKZ's contribution:

- CMAP-DeepBKZ can share *multiple* short vectors as block to accelerate the reduction process in every solver. In CMAP-DeepBKZ, each solver periodically sends its short basis vectors to a container of a supervisor. In contrast, the supervisor distributes short lattice vectors stored in its container to all solvers. Thus every solver can share short lattice vectors with the other solvers by communicating only with the supervisor.
- As the number of shared vectors increases, the reduction process can accelerate in every solver, but the randomness of the solver's bases might be lost. Therefore we propose a method to quantify the similarity of lattice bases using metrics for Grassmann manifolds (e.g., see [BG73; GVL96] for Grassmann metrics). Using the method, we verify by experiments the randomness of output bases of our parallel reduction algorithm in CMAP-DeepBKZ.
- We demonstrate the performance and the scalability of CMAP-DeepBKZ by large-scale experiments using up to 103,680 cores. Specifically, we evaluate how the quality of an output basis of our parallel algorithm changes, 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 [Sch+10] in dimensions up to around 130.

1.3 Related work

We summarize studies and software for the parallelization of lattice algorithms. Applications of high-performance computing to cryptanalysis for RSA and ECDSA are summarized [Jou12].

Divide and conquer Since the ENUM algorithm represents a search space as a depth-first tree structure, it is easy to divide the search space completely. This divide-and-conquer method divides the enumeration tree into sub-trees, and each search process is performed on different sub-trees and collects the results [DS10; Her+10; Kuo+11]. It has also been proposed to perform depth-first search in parallel on GPUs [Her+10] or FPGAs [Det+10]. Equalizing the size of the tasks in each search process can be achieved by creating a large number of tasks consisting of small sub-trees, but this rapidly increases the communication cost.

Task parallelization Another parallelization approach has been pursued by randomization [Kuo+11; BBK19]. Applying unimodular transformation to the basis vectors does not change the lattice but alters the enumeration tree. Hence, a parallel search can be conducted on the bases obtained by applying randomly generated unimodular matrices to the basis. In other words, while the divide-and-conquer parallel strategy targets a single enumeration tree, this randomization strategy searches multiple enumeration trees in parallel. Also, the *pruning technique* of the search tree [GNR10] can be effectively used for this parallel strategy. Instead of losing the guarantee that the shortest vector will be found, the number of nodes in the tree can be significantly reduced by the pruning technique. This property also serves to reduce the duplication of search in the randomized search tree. Before searching pruned enumeration trees, lattice basis reduction algorithms are performed to reduce the size of enumeration trees. [Kuo+11] creates SVP instances by randomization and performs lattice basis reduction and parallel ENUM independently on CPU or GPU using cloud computing. [BBK19] presented a shared-memory parallelized system based on randomization and extreme pruning of [GNR10]. However, it reports the runtime of solving exact-SVP for dimensions up to at most 100 over quad-socket Intel E7-4890 v2 CPUs (60 cores).

Data centralized parallelization Sieve-based algorithms utilize the large number of lattice vectors collected in a centralized place for the dominant part of the computation. Search processes perform nearly (if not completely) independent calculations to take advantage of the randomness in sampling. This scheme is suitable for shared memory systems where the memory is acceptable by all running threads, and concurrent accesses are handled explicitly. In 2019, Albrecht et al. [Alb+19] provided the General Sieve Kernel, abbreviated as G6K, that supports a variety of lattice basis reduction using advanced sieve algorithms. For BKZ with G6K, we can select a sieve algorithm to run as a core exact-SVP oracle in local block lattices. G6K adopts a multi-thread parallelization with highly optimized implementation for core sieve algorithms in high-dimensional lattices. In 2021, a GPU implementation was provided in [DSW21] for advanced sieve algorithms inside G6K to break high-dimensional instances in the Darmstadt SVP challenge (cf., see [PSZ21] for a GPU implementation of enumeration). In 2018, Teruya et al. [TKH18] proposed a massive parallelization for random sampling. In their system, basis vectors except the last few vectors are stored in global storage and shared with all processes in distributed computing platforms. Each process performs random sampling independently on its basis and competes to reduce the basis using vectors in the global storage. A synchronization processing is required only for storing and loading basis vectors between each process and the global storage.

Task parallelization with small data communication In 2020, a distributed and asynchronous parallel reduction algorithm was first developed in [Tat+20], which is called MAP-SVP (MAssively Parallel solver for SVP). It was built on the Ubiquity Generator (UG) framework [Ug], a generic framework for branch-and-band algorithms, to parallelize a reduction algorithm based on randomization that generates different bases of the same lattice by a 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, but it enables to share a shortest basis vector with all solvers to accelerate the reduction process of every solver. Above other parallelization methods have been implemented in single-program and multiple-data (SPMD) style. Besides, this parallelization is multi-program and multiple-data (MPMD) style, and the data, lattice vectors, are aggregated into a single control process. The performance and scalability of MAP-SVP were reported in [Tat+20, Section V] by using up to 100,032 cores for solving several instances of the Darmstadt SVP challenge [Sch+10].

1.4 Structure of this thesis

In Chapter 2, we introduce some definitions of lattice that are used throughout this thesis. These include representative lattice problems and their relationships. Chapter 3 describes basic algorithms for solving SVP that has various motivations and principles. We also explain the properties of each algorithm and how to use them to benefit from parallel computing. These algorithms were used to test the performance and flexibility of our framework. In Chapter 4, we introduce our framework for the massive parallelization of SVP algorithms. Our framework includes a new parallel scheme for lattice problems, where different algorithms are heterogeneously executed in parallel with information sharing. A design of our framework and some implementation techniques are also presented in this chapter. We demonstrate the performance using the basic algorithm on up to 103,680 cores in large-scale experiments. In Chapter 5, we present the first parallel solver using our framework, which parallelizes the lattice basis reduction by fully exploiting the features of our framework. The lattice basis reductions are accelerated by sharing short lattice vectors in the basis as a block. However, there is a trade-off between randomness and the amount of shared information. To quantify the randomness of lattice basis reduction, we also propose a novel metric using the Grassmann manifold. This metric is used for parameter tuning to benefit from the parallelization fully. In addition, we provide in-depth analyses of our solver's quality of output by sharing information and using large-scale computer platforms up to 103,680 cores.

In Appendix A, we give solutions of the SVP challenge found by our solver, including new records of the hall of frame of the SVP challenge. In Appendix B, we give a well-reduced lattice basis of the SVP challenge found by our solver.

Chapter 2

Preliminaries

In this chapter, we will introduce the background of lattices. We begin with the basic definitions of lattices, their properties, and the main lattice problems.

2.1 Lattices and their bases

Definition 2.1.1 (Lattice and lattice basis) *For integers $n \geq m \geq 1$, let $\mathbf{b}_1, \ldots, \mathbf{b}_m$ be m linearly independent vectors in $\mathbb{R}^n$. A* lattice *L is the set of all* integral *linear combinations of the $\mathbf{b}_i$'s. In other words, we have,*

$$L = \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_m) := \left\{ \sum_{i=1}^{m} v_i \mathbf{b}_i : v_i \in \mathbb{Z} \ (1 \leq \forall i \leq n) \right\}. \tag{2.1}$$

Besides, we call $\mathbf{B}$ a basis of lattice L when B consists of $\mathbf{b}_1, \ldots, \mathbf{b}_m$ vectors span the lattice $L = \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_m)$.

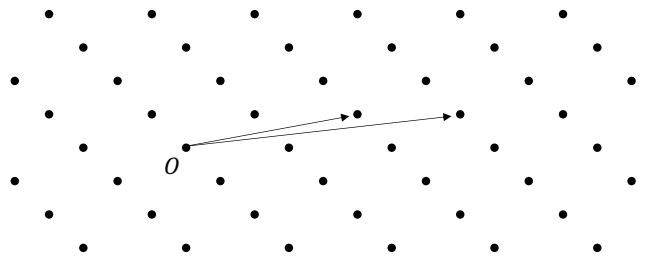

FIGURE 2.1: A lattice in $\mathbb{R}^2$ and their basis vectors

In this thesis, we denote lattice basis $\mathbf{B}$ as a matrix consists of $\mathbf{b}_1, \ldots, \mathbf{b}_m$ as column vectors in as follows

$$\mathbf{B} = \begin{bmatrix} \longleftarrow & \mathbf{b}_1 & \longrightarrow \\ \longleftarrow & \mathbf{b}_2 & \longrightarrow \\ & \vdots & \\ \longleftarrow & \mathbf{b}_m & \longrightarrow \end{bmatrix}.$$

Every lattice has infinitely many bases when $n, m \geq 2$; if two bases $\mathbf{B}_1$ and $\mathbf{B}_2$ span the same lattice, then there exists an $n \times n$ unimodular matrix $\mathbf{U}$ satisfying $\mathbf{B}_1 = \mathbf{B}_2\mathbf{U}$ (An integral square matrix with determinant ± 1 is called unimodular). Any elementary row operation of matrix is represented unimodular matrix, therefore for any basis $\mathbf{B}$ of a lattice, elementary row operations for basis $\mathbf{B}$ can not change the lattice $\mathcal{L}(\mathbf{B})$.

Definition 2.1.2 (Volume) *The* volume *of lattice $L = \mathcal{L}(\mathbf{B})$ is defined as*

$$\mathrm{vol}(L) = \sqrt{\det(\mathbf{B}\mathbf{B}^T)}.$$

Especially, when $n = m$ for a lattice basis $\mathbf{B} \in \mathbb{R}^{n \times m}$, we have $\mathrm{vol}(L) = \det(\mathbf{B})$. The volume of lattice L is independent of the choice of bases of L. It is the volume of the parallelepiped spanned by the vectors $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_m)$ in geometrically.

Definition 2.1.3 (Gram-Schmidt orthogonalization) *The* Gram-Schmidt orthogonalization *for a basis* $\mathbf{B}$ *is the orthogonal family* $\mathbf{B}^* = (\mathbf{b}_1^*, \ldots, \mathbf{b}_m^*)$*, recursively defined by* $\mathbf{b}_1^* = \mathbf{b}_1$ *and for* $2 \leq i \leq m$

$$\mathbf{b}_i^* = \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{ij} \mathbf{b}_j^* \text{ with } \mu_{ij} = \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\|\mathbf{b}_j^*\|^2} \ (j < i). \tag{2.2}$$

We call $\boldsymbol{\mu} = (\mu_{ij})$ the Gram-Schmidt orthogonalization coefficient matrix, where $\mu_{ij} = 0$ for all $i < j$ and $\mu_{kk} = 1$ for all $1 \leq k \leq m$. Then we have $\mathbf{B} = \mathbf{B}^* \boldsymbol{\mu}$, and thus

$$\mathrm{vol}(L) = \prod_{i=1}^{m} \|\mathbf{b}_i^*\|.$$

The Gram-Schmidt orthogonalization $\mathbf{B}^*$ is not unique of lattice L and this depends on the basis $\mathbf{B}$ and the order of the vectors of it.

Definition 2.1.4 (Orthogonal projection) *Let* π_ℓ *denote the orthogonal projection onto the orthogonal complement of the* $\mathbb{R}$*-vector space* $\langle \mathbf{b}_1, \ldots, \mathbf{b}_{\ell-1} \rangle_{\mathbb{R}}$ *defined by*

$$\pi_\ell : \mathbb{R}^n \longrightarrow \langle \mathbf{b}_1, \ldots, \mathbf{b}_{\ell-1} \rangle_{\mathbb{R}}^{\perp} = \langle \mathbf{b}_\ell^*, \ldots, \mathbf{b}_m^* \rangle_{\mathbb{R}},$$
$$\pi_\ell(\mathbf{x}) = \sum_{i=\ell}^{m} \frac{\langle \mathbf{x}, \mathbf{b}_i^* \rangle}{\|\mathbf{b}_i^*\|^2} \mathbf{b}_i^* \text{ for } \mathbf{x} \in \mathbb{R}^n.$$

Note that this projection map π_ℓ depends on the basis. We set $\pi_1 = \mathrm{id}$ (the identity map) for convenience. If $i \geq \ell$ then we have $\pi_\ell(\mathbf{b}_i) = \sum_{j=l}^{i} \mu_{ij} \mathbf{b}_j^*$ else $\pi_\ell(\mathbf{b}_i) = \mathbf{0}$.

Definition 2.1.5 (Projected lattice) *The lattice in* $\mathbb{R}^n$ *spanned by* $\pi_k(\mathbf{b}_k), \ldots, \pi_k(\mathbf{b}_d)$ *is called a* projected lattice *of* L*, denoted by* $\pi_k(L)$*.*

For the convenience of notation, we use $\mathbf{B}_{[i,j]}$ to mean the basis consisting of projected lattice vectors,

$$\mathbf{B}_{[i,j]} := \left(\pi_i\left(\mathbf{b}_i\right), \ldots, \pi_i\left(\mathbf{b}_j\right)\right),$$

and we use $L_{[i,j]}$ a lattice generated from $\mathbf{B}_{[i,j]}$,

$$L_{[i,j]} := \mathcal{L}\left(\mathbf{B}_{[i,j]}\right).$$

The projected lattice $L_{[i,j]}$ has dimension $j - i + 1$ and

$$\mathrm{vol}\left(L_{[i,j]}\right) = \prod_{k=i}^{j} \|\mathbf{b}_k^*\|$$

since the Gram-Schmidt orthogonalization of $(\pi_i(\mathbf{b}_i), \ldots, \pi_i(\mathbf{b}_j))$ is given by $\mathbf{b}_i^*, \ldots, \mathbf{b}_j^*$. Note that any projected lattice depends on a basis $\mathbf{B}$ of L.

Definition 2.1.6 (Successive minima) *For* $1 \leq k \leq n$*, the* k*-th* successive minimum *of a* m*-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 lattice L.

Definition 2.1.7 (Hermite Constant) *Let* $\mathscr{L}_m$ *be the set of* m*-dimensional lattice. Then, the* m*-dimensional* Hermite constant γ_m *is defined as*

$$\gamma_m := \max_{L \in \mathscr{L}_m} \frac{\lambda_1(L)^2}{\mathrm{vol}(L)^{2/m}} \tag{2.3}$$

This is also used for the upper bound analysis of the shortest vector norm in output basis by lattice basis reduction algorithm.

Gaussian Heuristic Given a lattice L of dimension n and a measurable set S in $\mathbb{R}^n$, the *Gaussian Heuristic* predicts that the number of vectors in $L \cap S$ is roughly equal to $\mathrm{vol}(S)/\mathrm{vol}(L)$. By applying to the ball C centered at the origin in $\mathbb{R}^n$ with radius $\lambda_1(L)$, it leads to

$$\frac{\mathrm{vol}(C)}{\mathrm{vol}(L)} \approx \#(L \cap C) \approx 1.$$

Using $\mathrm{vol}(C) = \omega_n \lambda_1(L)^n$, where ω_n denotes the volume of the unit ball in $\mathbb{R}^n$, then we have

$$\lambda_1(L) \approx \omega_n^{-\frac{1}{n}} \mathrm{vol}(L)^{\frac{1}{n}}.$$

We denote this heuristic estimation of $\lambda_1(L)$ as follows.

$$\mathrm{GH}(L) := \omega_n^{-\frac{1}{n}} \mathrm{vol}(L)^{\frac{1}{n}} \sim \sqrt{\frac{n}{2\pi e}} \mathrm{vol}(L)^{\frac{1}{n}}. \tag{2.4}$$

Approximation $\omega_n^{-\frac{1}{n}} \sim \sqrt{\frac{n}{2\pi e}}$ is derived from Stirling's approximation. $\mathrm{GH}(L)$ is only a heuristic, but it roughly holds for random lattices [GM03] in high dimensions such as $n \geq 50$.

2.2 Lattices Problems

Lattice problems are algorithmic problems that involve lattices. Among lattice problems, the SVP, CVP and their variants are fundamental importance.

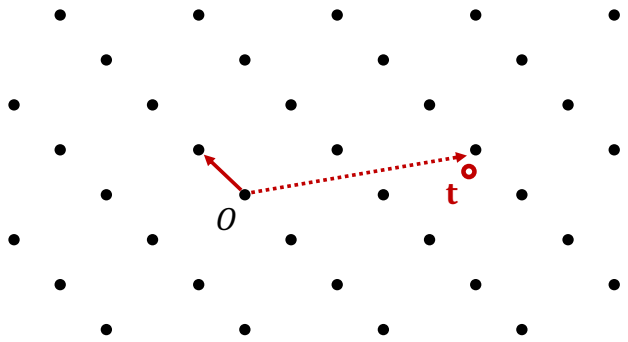

FIGURE 2.2: Example of solutions of SVP (in Definition 2.2.1) and CVP (in Definition 2.2.6) for 2-dimensional lattice L; a solid vector represents a shortest vector in L , and break vector represents a closest vector in L for a vector $\mathbf{t}$

Definition 2.2.1 (Shortest Vector Problem (SVP)) *Find the shortest non-zero vector with respect to the ℓ_2-norm in the lattice L. In the form of optimization,*

$$\min_{\mathbf{v}} \|\mathbf{v}\| \text{ such that } \mathbf{v} \in L \setminus \{\mathbf{0}\}.$$

SVP is a discrete combination optimization problem for finding x_i's in (2.1) and is shown to be NP-hard under randomized reductions [Ajt96]. (That is, a probabilistic Turing-machine exists that reduces any problem in NP to SVP instances in polynomial-time.)

The followings are evaluation metrics for the found lattice vector and basis.

Definition 2.2.2 (Approximation Factor) *For a vector $\mathbf{v} \in L$, the value $\|\mathbf{v}\|/\mathrm{GH}(L)$ is called the* approximation factor *of $\mathbf{v}$. Similarly, for a basis matrix $\mathbf{B}$, the value $\min_{1 \leq i \leq n} \|\mathbf{b}_i\|/\mathrm{GH}(\mathcal{L}(\mathbf{B}))$ is called the approximation factor of $\mathbf{B}$.*

Definition 2.2.3 (Hermite Factor) *For a vector* $\mathbf{v} \in L$*, the value* $\|\mathbf{v}\|/\mathrm{vol}(L)^{1/n}$ *is called the* Hermite factor *of* $\mathbf{v}$*. Similarly, for a basis matrix* $\mathbf{B}$*, the value* $\min_{1\leq i\leq n}\|\mathbf{b}_i\|/\mathrm{vol}(\mathcal{L}(\mathbf{B}))^{1/n}$ *is called the Hermite factor of* $\mathbf{B}$*.*

Based on these metrics, an approximate variant of SVP is defined:

Definition 2.2.4 (Approximate Shortest Vector Problem (ASVP)) *Given a lattice* L *and an approximation factor* $\gamma > 0$*,*

$$\textit{find } \mathbf{v} \in L \setminus \{\mathbf{0}\} \textit{ such that } \|\mathbf{v}\| \leq \gamma \cdot \lambda_1(L).$$

ASVP is exactly SVP when $\gamma = 1$. If $\gamma < \sqrt{2}$, ASVP becomes NP-hard [Mic01].

Definition 2.2.5 (Hermite Shortest Vector Problem (HSVP)) *Given a lattice* L *and an approximation factor* $\gamma > 0$*,*

$$\textit{find } \mathbf{v} \in L \setminus \{\mathbf{0}\} \textit{ such that } \|\mathbf{v}\| \leq \gamma \cdot \mathrm{vol}(\mathcal{L}(\mathbf{B}))^{1/n}.$$

Another important lattice problem is:

Definition 2.2.6 (Closest Vector Problem (CVP)) *Given a a lattice* L *and a target vector* $\mathbf{t}$*, find a vector in* L *that is closest to* $\mathbf{t}$*. In the form of optimization,*

$$\min_{\mathbf{v}} \|\mathbf{v} - \mathbf{t}\| \textit{ such that } \mathbf{v} \in L$$

From a practical point of view, however, both problems are considered equally hard due to Kannan's embedding technique [Kan87] that can transform CVP into SVP. The main idea of Kannan's embedding technique is to define a lattice $\tilde{L}$ containing a short vector $\mathbf{e} = \mathbf{t} - \mathbf{s}$, where $\mathbf{s}$ is the optimal solution of CVP. We define a lattice $\tilde{L}$ as $\mathcal{L}((\mathbf{b}_1, 0), \ldots, (\mathbf{b}_m, 0), (\mathbf{t}, M))$ where $0 < M \in \mathbb{R}$. Then, by solving SVP on $\tilde{L}$, we can obtain vector $\mathbf{e}$ and $\mathbf{s}$ as $\mathbf{t} - \mathbf{e}$.

A particular case of CVP that we will use later in this thesis is

Definition 2.2.7 (Bounded Distance Decoding (BDD)) *Given a lattice* L *and a target vector* $\mathbf{t}$ *within distance* $\alpha\lambda_1(L)$ *of* $L = \mathcal{L}(\mathbf{B})$ *for a constant* $0 < \alpha \leq \frac{1}{2}$*, find a vector in* L *closest to* $\mathbf{t}$*.*

There are other essential lattice problems related to the security of modern lattice-based cryptosystems, such as the learning with errors and NTRU problems (e.g., see [Pei16]). Most lattice problems can be reduced to SVP or CVP, so SVP and CVP are fundamental. As Kannan's embedding transforms CVP into SVP, We focus on SVP in this thesis to simplify the narrative. However, the proposed framework in Chapter 4 apply to other lattice problems.

Chapter 3

Algorithms for Shortest Vector Problem

This chapter introduces primary families of lattice algorithms for solving the shortest vector problem (SVP). We can categorize these algorithms into two types. One is an exact-SVP algorithm, and the other is an approximate-SVP algorithm. These algorithms are not independent but are closely related. For example, the approximate-SVP algorithm uses the exact-SVP algorithm internally, and the output of the approximate-SVP algorithm can be used for the exact SVP algorithm. We introduce Enumeration and sieve as the exact-SVP algorithm and lattice basis reduction as the approximate-SVP algorithm. These algorithms are used in our parallel frameworks and software that will be introduced in later chapters.

3.1 Enumeration

Enumeration (ENUM) algorithm is a deterministic algorithm solving SVP exactly. For an SVP instance of dimension m, the time complexity is $2^{O(m^2)}$, but the space complexity is a polynomial in m. Given a basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_m\}$ of a lattice L, ENUM is based on a depth-first tree search for an integer combination $(v_1, \ldots, v_m)$ such that $\mathbf{v} = v_1\mathbf{b}_1 + \cdots + v_m\mathbf{b}_m$ has the shortest norm in $L \setminus \{\mathbf{0}\}$.

With the Gram-Schmidt information (2.2), the target vector can be written as

$$\mathbf{v} = \sum_{i=1}^{m} v_i \left(\mathbf{b}_i^* + \sum_{j=1}^{i-1} \mu_{ij}\mathbf{b}_j^* \right) = \sum_{j=1}^{m} \left(v_j + \sum_{i=j+1}^{m} \mu_{ij} v_i \right) \mathbf{b}_j^*$$

By the orthogonality of $\mathbf{b}_i^*$'s, the projected vector $\pi_k(\mathbf{v})$ has length

$$\rho_k = \|\pi_k(\mathbf{v})\|^2 = \sum_{j=k}^{m} \left(v_j + \sum_{i=j+1}^{m} \mu_{ij} v_i \right)^2 \|\mathbf{b}_j^*\|^2 \quad (1 \le k \le m).$$

Given a search radius $R > 0$, ENUM constructs an enumeration tree of depth m, whose nodes at depth $m - k + 1$ correspond to the set of all vectors in projected lattices $\pi_k(L)$ with a maximum length of R. The key observation is that if a shortest vector satisfies $\|\mathbf{v}\| < R$, its projections satisfy $\|\pi_k(\mathbf{v})\|^2 \le R^2$ for all $1 \le k \le m$ since $\|\pi_k(\mathbf{v})\|^2 \le \|\mathbf{v}\|^2$; hence, it appears as a leaf of the tree. These m inequalities provide an efficient enumeration of the tree. The total number of nodes to be searched can be estimated using the Gaussian Heuristic for each projected lattice as $\sum_{\ell=1}^{m} H_\ell$, where

$$H_\ell := \frac{R^\ell \omega_\ell}{\mathrm{vol}(\pi_{m+1-\ell}(L))} = \frac{R^\ell \omega_\ell}{\prod_{i=m+1-\ell}^{m} \|\mathbf{b}_i^*\|} \quad (1 \le \ell \le m), \tag{3.1}$$

and ω_l denotes the volume of the unit ball in $\mathbb{R}^l$. Therefore, it is crucial to choose a good R, which is sufficiently small but larger than he shortest norm. One useful strategy is pruning [GNR10] where a smaller tree is built by replacing the inequalities $\|\pi_k(\mathbf{v})\|^2 \le R^2$ by $\|\pi_k(\mathbf{v})\|^2 \le R_{m+1-k}^2$ with a shorter radii $R_1 \le \cdots \le R_m = R$ at each depth defined by a

Algorithm 1 ENUM [GNR10]

1: **procedure** ENUM($\mathbf{B}, \mathbf{R}$)
2: ▷ $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_m)$: basis of a lattice L, $\mathbf{R} = (R_1, \ldots, R_m)$: R_k is a radius of depth-k projected lattice $\pi_{n-k+1}(\mathcal{L}(\mathbf{B}))$
3: Set $\boldsymbol{\rho}, \mathbf{r}, \mathbf{v}, \mathbf{c}$ and $\mathbf{w}$ to zero array, whose size is $m+1, m+1, m, m$ and m, respectively;
4: Set σ to zero matrix, whose size is $(m+1) \times m$;
5: $r_2 \leftarrow 1$; $v_1 \leftarrow 1$; $\ell \leftarrow 1$;
6: **while** true **do**
7: $\rho_k \leftarrow \rho_{k+1} + (v_k - c_k)^2 \|\mathbf{b}_k^*\|^2$; ▷ $\rho_k = \|\pi_k(\mathbf{v})\|^2$
8: **if** $\rho_k \leq R_{m+1-k}^2$ **then**
9: **if** $k = 1$ **then return** $\mathbf{v} = \sum_{k=1}^{m} v_k \mathbf{b}_k$;
10: ▷ Find $\mathbf{v}$ such that $\|\pi_k(\mathbf{v})\|^2 \leq R_{m+1-k}^2$ for all k
11: **end if**
12: $k \leftarrow k - 1$; $r_k \leftarrow \max(r_k, r_{k+1})$;
13: **for** $i = r_{k+1}$ down to $k+1$ **do**
14: $\sigma_{i,k} \leftarrow \sigma_{i+1,k} + \mu_{ik} v_i$;
15: **end for**
16: $c_k \leftarrow -\sigma_{k+1,k}$; $v_k \leftarrow \lfloor c_k \rceil$; $w_k \leftarrow 1$;
17: **else**
18: $k \leftarrow k + 1$;
19: **if** $k = m + 1$ **then return 0**; ▷ Finish search
20: **else** ▷ Go to the successor
21: $r_{k-1} \leftarrow k$;
22: **if** $k \geq \ell$ **then**
23: $\ell \leftarrow k$; $v_k \leftarrow v_k + 1$;
24: **else**
25: **if** $v_k > c_k$ **then**
26: $v_k \leftarrow v_k - w_k$;
27: **else**
28: $v_k \leftarrow v_k + w_k$;
29: **end if**
30: **end if**
31: $w_k \leftarrow w_k + 1$;
32: **end if**
33: **end if**
34: **end while**
35: **end procedure**

pruning strategy. This is a probabilistic method because it is not certain that $\mathbf{v}$ can be found in this pruned tree.

The description in pseudo-code of ENUM is given in Algorithm 1. In this algorithm, there is no element of randomness, and the nodes are traversed deterministically. ENUM can obtain the shortest vector by setting radius $R_1 = \cdots = R_m = R$ to the norm of the vector currently known and continuing the search without returning in line 9. In addition, since this algorithm works well even if the parameter R_i are updated while the algorithm is running, we can reduce the number of nodes of the enumeration tree without losing the ability to find the shortest vector by updating the parameters R_i with the norm of $\mathbf{v}$ in line 9. The size of the enumeration tree is determined by the input lattice basis and the radius parameters R_i.

Using a basis which is an output of lattice reduction algorithm (described in Section 3.3), the number of nodes in enumeration tree becomes generally smaller. It was pointed out in [GN08] that the approximation

$$\|\mathbf{b}_i^*\| / \|\mathbf{b}_{i+1}^*\| \approx q \quad (1 \leq i \leq m-1)$$

holds for the Gram-Schmidt coefficients $\mathbf{b}_1^*, \ldots, \mathbf{b}_m^*$ of the lattice basis output by the lattice

reduction algorithm. In addition, from definition of Hermite constant (2.3), the upper bound on the shortest vector norm of m-dimensional lattice is given by $\sqrt{\gamma_m}\mathrm{vol}(L)^{\frac{1}{m}}$. From Gram-Schmidt coefficient's property and setting the radius R to $\sqrt{\gamma_m}\mathrm{vol}(L)^{\frac{1}{m}}$ which is optimal in the worst case, then H_ℓ in (3.1) are approximated as follows [GNR10];

$$H_\ell \approx q^{(m-\ell)\ell/2} V_\ell(\sqrt{\gamma_m}).$$

γ_m is Hermite constant in (2.3), and from this definitions, the upper bound shortest vector of lattice Therefore, if as q becomes smaller, the number of nodes in the enumeration tree becomes smaller. It is known that q can be reduced by transforming the lattice basis with a stronger reduction algorithm. In other words, we can make the ENUM algorithm work under better conditions by preprocessing of reduction algorithms.

Using as a sampler of short lattice vector In Algorithm 1, we can search all the lattice vectors $\mathbf{v}$ satisfying $\|\pi_k(\mathbf{v})\|^2 \leq R^2_{m-k+1}$, especially $\|\mathbf{v}\| = \|\pi_1(\mathbf{v})\| < R_m$, by continuing the search without returning in line 9. The number of vectors satisfying the condition can be estimated from basis $\mathbf{B}$ and radius R. Since an enumeration tree depends on the input basis, it is not easy to switch the basis in the middle of the algorithm. If we want to perform ENUM from another basis, we must terminate the search and start ENUM using another basis. The memory usage of this algorithm is minimal and does not increase during the search.

3.2 Sieve

Sieve algorithm has a better asymptotic runtime than enumeration, but it requires exponential space $2^{\Theta(n)}$. The first algorithm of this kind is the randomized sieve algorithm proposed by Ajtai, Kumar and Sivakumar (AKS) [AKS01]. It outputs a shortest lattice vector with overwhelming probability, and its asymptotic complexity is much better than deterministic enumeration algorithms with $2^{O(n^2)}$ time complexity. The idea is that given a lattice L of dimension n, consider a ball S centered at the origin and of radius r with $\lambda_1(L) \leq r \leq O(\lambda_1(L))$. Then $\#(L \cap S) = 2^{O(n)}$ according to the Gaussian Heuristic. If we could perform an exhaustive search for all vectors in $L \cap S$, we could find a shortest lattice vector within $2^{O(n)}$ polynomial-time operations. In contrast, the AKS algorithm performs a randomized sampling of $L \cap S$. If it was uniformly sampled over $L \cap S$, a short lattice vector would be included in N samples with probability close to 1 for $N \gg \#(L \cap S)$. It can be also shown that there exists a vector $\mathbf{w} \in L \cap S$ such that $\mathbf{w}$ and $\mathbf{w} + \mathbf{s}$ can be sampled with non-zero probability for some shortest lattice vector $\mathbf{s}$. Thus a shortest lattice vector is obtained by computing a shortest difference of any pairs of the N sampled vectors in $L \cap S$. There are various implementations of sieve algorithms that differ mainly in how to sample lattice vectors, such as ListSieve and GaussSieve [MV10]. Similarly to ENUM, the choice of R is crucial to the sieve.

GaussSieve Here we describe the *GaussSieve* algorithm, which is incorporated into our framework for testing, and a pseudo-code is shown in Algorithm 2. In GaussSieve, vectors are sampled sequentially and stored in a *List*. Then, to keep that any vector pair of $(\mathbf{v}, \mathbf{p})$ in the list are satisfied the pairwise reduced condition, $\min(\|\mathbf{v} \pm \mathbf{p}\|) \geq \max(\|\mathbf{v}\|, \|\mathbf{p}\|)$, we reduce the sampled vector using vectors in the list. Simultaneously, the vectors in the List are also reduced by the reduced sampled vector. As a result, vectors that do not satisfy the pairwise reduced condition are moved to a *Stack* and them will use instead of sampling. If the reduced vectors collide with the vectors in List c times, algorithm terminates. We use a priority queue data container to List and Stack. Its norm sorts the vectors in the priority queue. Sampling is executed by Klein's randomized rounding algorithm [Kle00].

Using as a sampler of short lattice vector The norm of the lattice vectors output by a Klein sampler is generally long, and at the beginning of the algorithm, when the number of vectors in a List is small, lattice vectors in a List are long. As the number of vectors in a List

Algorithm 2 GaussSieve [MV10]

```
1: procedure GaussSieve(B, c)
2:             ▷ B = (b_1, ..., b_m): basis of a lattice L, c: the maximum number of collision
3:     List ← {0}; Stack ← {}; K ← 0;
4:     while K < c do
5:         if Stack is empty then
6:             Sample v using klein sampler with B;
7:         else
8:             Pop v from Stack;
9:         end if
10:        v ← GaussReduce(v, List, Stack);
11:        if v = 0 then
12:            K ← K + 1;
13:        else
14:            Push v into List;
15:        end if
16:    end while
17: end procedure
18: procedure GaussReduce(v, List, Stack)
19:    while ∃p ∈ L such that ‖p‖ ≤ ‖v‖ ∧ ‖v − p‖ ≤ ‖v‖ do
20:        v ← v − p;                                   ▷ Reduce v using p
21:    end while
22:    while ∃p ∈ L such that ‖p‖ > ‖v‖ ∧ ‖p − v‖ ≤ ‖p‖ do
23:        Pop p from List;
24:        Push p into Stack;
25:    end while
26: end procedure
```

increases, the sampled vectors become strongly reduced, and the number of short vectors in a List increases. In other words, the algorithm generates shorter vectors as the algorithm proceeds, and the sampler's performance improves. We also can interfere from the outside of the algorithm by adding lattice vectors to Stack because Stack is managed independently from the pairwise reduction condition in List.

3.3 Lattice basis reduction

Reduction algorithms find not necessarily shortest lattice vectors, but they are much faster than exact-SVP algorithms such as enumeration and sieve (see [Ngu09; Yas21] for a survey). Given a basis of a lattice, the goal of lattice basis reduction is to find a new basis of the same lattice consisting of nearly orthogonal and relatively short vectors (See Figure. 3.1). Most lattice problems become easier to solve with such a reduced basis. The Lenstra-Lenstra-Lovász (LLL) algorithm [LLL82] is the most celebrated algorithm, and its blockwise generalization is the block Korkine-Zolotarev (BKZ) algorithm [SE94]. Recently, efficient variants of BKZ such as BKZ 2.0 [CN11] are implemented in software libraries (e.g., fplll library [The16]), and they are used to estimate the security level of lattice-based schemes (e.g., see [AD21; Alb+18]).

3.3.1 LLL reduction

Here, we first introduce the size-reduction algorithm [Her50], which is the basic component of various lattice basis reduction algorithms.

Definition 3.3.1 (Size-reduction) *a matrix* $\mathbf{B} \in \mathbb{R}^{m \times n}$ *is called size-reduced, if its satisfies:*

$$|\mu_{ij}| \leq \frac{1}{2} \quad (1 \leq j \leq i \leq m).$$

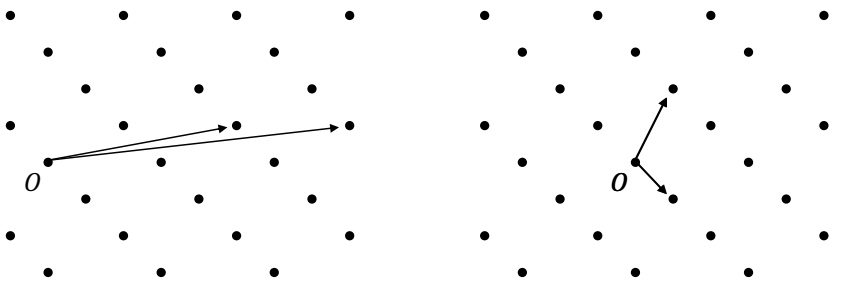

FIGURE 3.1: An example of lattice reduction: Left is lattice basis before lattice reduction, right is that after lattice reduction.

Algorithm 3 Size-reduction algorithm [Her50]

```
1: procedure SizeReduction(B)                ▷ B = (b_1, ..., b_m): basis of a lattice L
2:     for i = 2 to m do
3:         for j = i − 1 down to 1 do
4:             SizeReduce(B, i, j);
5:         end for
6:     end for
7: end procedure
8: procedure SizeReduce(B, i, j)
9:     if |μ_ij| > 1/2 then
10:        q ← ⌊μ_ij⌉;
11:        b_i ← b_i − q b_j;
12:        for ℓ = 1 to j do
13:            μ_iℓ ← μ_iℓ − q μ_jℓ;
14:        end for
15:    end if
16: end procedure
```

We can obtain the size-reduced basis by Algorithm 3. Since $\|\mu_{ij}\|$ is calculated from the inner product of $\mathbf{b}_i^*$ and $\mathbf{b}_j^*$, a smaller value indicates that $\mathbf{b}_i^*$ and $\mathbf{b}_j^*$ are closer to orthogonal.

Definition 3.3.2 (δ-LLL-reduction) *For $\frac{1}{4} < \delta < 1$, a matrix $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_m) \in \mathbb{R}^{m \times n}$ is called δ-LLL reduced, if it is size-reduced and satisfies the Lovász condition:*

$$\delta \|\pi_{i-1}(\mathbf{b}_{i-1})\| \leq \|\pi_{i-1}(\mathbf{b}_i)\|^2 \quad (1 \leq i < n).$$

This Lovász condition is equal to the following condition.

$$\|\mathbf{b}_i^*\| \geq \left(\delta - \mu_{i,i-1}^2\right) \|\mathbf{b}_{i-1}^*\|.$$

For a δ-LLL-reduced basis $\mathbf{B}$, it holds both

$$\|\mathbf{b}_1\| \leq \alpha^{\frac{m-1}{2}} \lambda_1(L), \text{ and}$$
$$\|\mathbf{b}_1\| \leq \alpha^{\frac{m-1}{4}} \mathrm{vol}(L)^{\frac{1}{m}}$$

for $L = \mathcal{L}(\mathbf{B})$ and $\alpha = 4/(4\delta - 1)$ (see [Bre11; Ngu09]). To find an LLL-reduced basis, the LLL algorithm [LLL82] calls size-reduction as a subroutine, and it also swaps adjacent basis vectors that do not satisfy Lovász' condition. The LLL algorithm has a complexity polynomial in m. In practice, the average approximation factor is smaller than this upper bound when using random lattices. Experiments conducted in [GN08] with a large number of random lattice bases show that in higher dimensions, on average, $\|\mathbf{b}_1\| \approx 1.021^m \mathrm{vol}(L)^{1/m}$.

In addition, MLLL algorithm [Poh87], which is a variant of LLL, can be get rid of the linear dependency of vectors. MLLL is used in the BKZ algorithm described below. Since

Algorithm 4 LLL algorithm [LLL82]

1: **procedure** LLL($\mathbf{B}$, δ)
2: ▷ $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_d)$: basis of a lattice L, δ: parameter of the Lovász condition
3: $B_i \leftarrow \|\mathbf{b}_i^*\|^2 (1 \le i \le n)$;
4: $k \leftarrow 2$;
5: **while** $k \le n$ **do**
6: **for** $j = k-1$ down to 1 **do**
7: SizeReduce($\mathbf{B}, k, j$);
8: **end for**
9: **end while**
10: **if** $B_k \le \left(\delta - \mu_{k,k-1}^2\right) B_{k-1}$ **then**
11: $k \leftarrow k+1$;
12: **else**
13: swap($\mathbf{B}, k, k-1$); ▷ Swap $\mathbf{b}_k$ and $\mathbf{b}_{k-1}$
14: $k \leftarrow \max\{k-1, 2\}$;
15: **end if**
16: **end procedure**

the MLLL algorithm is a generalization of the LLL algorithm, we will refer to MLLL as LLL in the following description. In other words, we assume that the MLLL algorithm works when a matrix that is not full rank is given as input to LLL.

As a generalization of LLL, *non-adjacent* basis vectors can be changed in LLL with deep insertions (DeepLLL) [SE94]; Given a basis $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ and a reduction parameter $\frac{1}{4} < \delta < 1$, we insert the k-th basis vector $\mathbf{b}_k$ before $\mathbf{b}_i$ as

$$\mathbf{B} \longleftarrow (\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}, \mathbf{b}_k, \mathbf{b}_i, \ldots, \mathbf{b}_{k-1}, \mathbf{b}_{k+1}, \ldots, \mathbf{b}_n) \tag{3.2}$$

for indexes $i < k$ such that $\|\pi_i(\mathbf{b}_k)\|^2 < \delta \|\mathbf{b}_i^*\|^2$, instead of swapping the neighboring basis vectors (at line 13 in Algorithm 4). This basis permutation is called 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.

Definition 3.3.3 (δ-DeepLLL-reduced) *We say a basis $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_m)$ δ-*DeepLLL-reduced *if it is size-reduced and $\delta \|\mathbf{b}_i^*\|^2 \le \|\pi_i(\mathbf{b}_k)\|^2$ for all $i < k$.*

For a δ-DeepLLL-reduced basis $\mathbf{B}$, it holds both

$$\|\mathbf{b}_1\| \le \sqrt{\alpha} \left(1 + \frac{\alpha}{4}\right)^{\frac{m-2}{2}} \lambda_1(L), \text{ and}$$
$$\|\mathbf{b}_1\| \le \alpha^{\frac{m-1}{2m}} \left(1 + \frac{\alpha}{4}\right)^{\frac{(m-1)(m-2)}{4m}} \mathrm{vol}(L)^{\frac{1}{m}}$$

for $L = \mathcal{L}(\mathbf{B})$ and $\alpha = \frac{4}{4\alpha - 1}$ (see [YY19]). These properties are better than LLL, but the complexity is no longer polynomial.

3.3.2 HKZ reduction

Hermite-Korkine-Zolotarev (HKZ) reduction has a more (ideal) strong reduction for lattice basis than LLL.

Definition 3.3.4 (HKZ-reduction) *A matrix $\mathbf{B} \in \mathbb{R}^{n \times n}$ is called HKZ-reduced, if it is size-reduced and satisfies:*

$$\|\mathbf{b}_i^*\| = \lambda_1\left(L_{[i,n]}\right) \quad (1 \le i \le n).$$

The HKZ-reduced basis has a smaller upper bound than LLL for the norm of $\mathbf{b}_1$.

Lemma 3.3.1 ([LLS90, Theorem 2.1]) *Let $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_m)$ be an HKZ-reduced basis of a lattice L. We have*

$$\frac{4}{i+3} \lambda_i(L)^2 \le \|\mathbf{b}_i\| \le \frac{i+3}{4} \lambda_i(L)^2 \quad (1 \le i \le m).$$

To obtain the HKZ-reduced basis, we need to solve the SVP on the projected lattice sequentially while incrementing i for $L_{[i,m]}$ from $i = 0$. However, since the cost of SVP increases exponentially with the dimension of lattice, it is tough to find the HKZ-reduced basis in practice.

3.3.3 BKZ reduction

The Blockwise Korkine-Zolotarev (BKZ) lattice reduction algorithm of Schnorr-Euchner [Sch87; SE94]. It generalizes the HKZ algorithm by introducing a blocksize $\beta > 2$. On the other hand, if $\beta = 1$, then BKZ reduction is equal to LLL reduction, and BKZ reduction can be said to be a generalization of LLL.

Definition 3.3.5 (β-BKZ reduced basis) *A matrix $\mathbf{B} \in \mathbb{R}^{n \times n}$ is called β-BKZ-reduced, if it is LLL-reduced and satisfies:*

$$\|\mathbf{b}_j^*\| = \lambda_1(L_{[j,k]}) \quad (1 \leq j \leq n),$$

where $k = \min(j + \beta - 1, d)$.

Note that $L_{[j,k]}$ is the projected lattice as $\mathcal{L}(B_{[j,k]})$, and $B_{[j,k]} = (\pi_j(\mathbf{b}_j), \ldots, \pi_j(\mathbf{b}_k))$.

For a β-BKZ-reduced basis $\mathbf{B}$, it holds

$$\|\mathbf{b}_1\| \leq \gamma_\beta^{\frac{d-1}{\beta-1}} \lambda_1(L),$$

where γ_β denotes Hermite's constant of dimension β [Sch92] (see [Ngu09] for Hermite's constants). A β-BKZ-reduced basis can be found by the BKZ algorithm [SE94], 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 β 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 $L_{[j,k]}$. Experimentally results in [GN08] shows $\|\mathbf{b}_1\| \approx 1.0128^m \mathrm{vol}(L)^{1/m}$ and $1.0109^m \mathrm{vol}(L)^{1/m}$ for blocksize $\beta = 20$ and 28, respectively, for high-dimensional lattice.

DeepBKZ It is an enhancement of BKZ proposed in [YY17] that uses DeepLLL as a subroutine in a BKZ framework (instead of LLL). We show a basic procedure of DeepBKZ in Algorithm 5 that calls enumeration as an exact-SVP algorithm in line 7. In practice, DeepBKZ can find shorter lattice vectors than BKZ in using the same blocksize β (see [YY17; YNY20] for their experimental results). Similarly to BKZ, the complexity of DeepBKZ depends on that of an exact-SVP algorithm (e.g., enumeration) in dimension β.

Using lattice basis reduction as a sampler of shortest lattice vector Although basis reduction does not aim to obtain the shortest lattice vector, it is experimentally known that it can find small lattice vectors whose norm is less than the theoretical upper bound ([BSW18] calls this phenomenon "head concave"). Also, the vectors in the lattice basis are frequently replaced during the processing of the algorithm. Therefore, we can sample short lattice vectors by fetching the vectors in the basis of the algorithm running at any timing. In addition, since the behavior of the basis reduction changes by randomization with unimodular matrices, it is also possible to sample short lattice vectors by repeating the basis randomization and the execution of (light) basis reduction.

3.4 Project-and-lift

The computational complexity of every known algorithm for SVP is exponential. A workaround is to work with a smaller dimensional lattice and lift its shortest vector to find a short vector in the original lattice. A straightforward but effective approach is to project the original basis vectors by π_k for some $1 \leq k < m$. First, find shortest vectors in the projected $(m - k + 1)$-dimensional lattice by, for example, ENUM or sieve, and lift them to the original lattice so

Algorithm 5 DeepBKZ [YY17]

procedure DeepBKZ($\mathbf{B}$, δ, β)
  ▷ $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_m)$: basis of a lattice L, δ: reduction parameter, β: blocksize
  $\mathbf{B} \leftarrow$ DeepLLL($\mathbf{B}, \delta$) ▷ DeepLLL-reduction for the input basis $\mathbf{B}$;
  $z \leftarrow 0$, $j \leftarrow 0$;
  while $z < m - 1$ **do**
    $j \leftarrow (j \pmod{m-1}) + 1$, $k \leftarrow \min(j + \beta - 1, m)$, $h \leftarrow \min(k + 1, m)$
    $\mathbf{v} \leftarrow$ ENUM($L_{[j,k]}$);
      ▷ Enumeration over $L_{[j,k]}$ to find $\mathbf{v} \in L$ satisfying $\|\pi_j(\mathbf{v})\| = \lambda_1(L_{[j,k]})$
    if $\|\pi_j(\mathbf{v})\| < \|\mathbf{b}_j^*\|$ **then**
      $z \leftarrow 0$, $(\mathbf{b}_1, \ldots, \mathbf{b}_h) \leftarrow$ LLL$((\mathbf{b}_1, \ldots, \mathbf{b}_{j-1}, \mathbf{v}, \mathbf{b}_j, \ldots, \mathbf{b}_h))$
        ▷ Remove the lsinear dependency by LLL after insertion of $\mathbf{v}$ at position j
    else
      $z \leftarrow z + 1$;
    end if
    DeepLLL$((\mathbf{b}_1, \ldots, \mathbf{b}_h), \delta)$;
      ▷ DeepLLL-reduction for the sub-basis $(\mathbf{b}_1, \ldots, \mathbf{b}_h)$ of the current basis $\mathbf{B}$
  end while
end procedure

that their projections by π_k coincides with the shortest vectors in the projected lattice. The latter lifting process is equivalent to BDD. In this manner, however, it is not guaranteed that a shortest vector will be found.

Sub-sieve Sub-sieve is proposed in [Duc18] which implements this idea using a sieve. Specifically, a sieve algorithm is performed in a projected lattice $\pi_k(L)$ to obtain a list of short lattice vectors:

$$D_{k,\tau} := \{\mathbf{0} \neq \mathbf{v} \in \pi_k(L) : \|\mathbf{v}\| \leq \tau \cdot \mathrm{GH}(\pi_k(L))\}$$

for a constant τ such as $\tau = \sqrt{\frac{4}{3}}$. In practice, k is chosen to be around $m - 30$ for high-dimensional lattices [Alb+19; DSW21]. Then, by Babai's algorithms [Bab86], the short vectors in the inverse image $\pi_k^{-1}(D_{k,\tau}) \subset L$ are enumerated. For a shortest non-zero vector $\mathbf{s}$ in L, we set d and τ so that the projected vector $\mathbf{s}_k := \pi_k(\mathbf{s})$ is included in the list $D_{k,\tau}$. By an exhaustive search over $D_{k,\tau}$, assume that $\mathbf{s}_k$ is known. Let $\mathbf{B}$ denote the basis matrix corresponding to $\{\mathbf{b}_1, \ldots, \mathbf{b}_m\}$. Write $\mathbf{s} = \mathbf{x}\mathbf{B}$ for some $\mathbf{x} \in \mathbb{Z}^n$, and split $\mathbf{x}$ as $(\mathbf{x}_1|\mathbf{x}_2)$ with $\mathbf{x}_1 \in \mathbb{Z}^{k-1}$ and $\mathbf{x}_2 \in \mathbb{Z}^{n-k+1}$. Since $\mathbf{s}_k = \pi_k(\mathbf{x}\mathbf{B}) = \mathbf{x}_2\mathbf{B}_k$, we know $\mathbf{x}_2$. Here $\mathbf{B}_k$ denotes the matrix whose rows are $\mathbf{b}_k, \ldots, \mathbf{b}_n$. We need to recover $\mathbf{x}_1$ so that the vector $\mathbf{s} = \mathbf{x}_1\mathbf{B}_1 + \mathbf{x}_2\mathbf{B}_2$ is the shortest in $L \setminus \{\mathbf{0}\}$, where we split $\mathbf{B}$ into two matrices $\mathbf{B}_1$ and $\mathbf{B}_2$. This is a BDD instance over the lattice spanned by the rows of $\mathbf{B}_1$ for the target vector $\mathbf{x}_2\mathbf{B}_2$.

Sub-ENUM We introduce sub-ENUM algorithm [Tat+21]. The first part is very similar to sub-sieve. An ENUM algorithm is performed in a projected lattice $\pi_k(L)$ to collect a lot of very short lattice vectors in $D_{k,\tau}$. We call this strategy child-ENUM. Then, instead of Babai's algorithms, an ENUM algorithm is again used to find a shortest vector for a k-dimensional lattice spanned by $\{\mathbf{b}_1, \ldots, \mathbf{b}_{k-1}, \mathbf{v}\}$ We call this root-ENUM. As described in Section 3.1, ENUM has complexity worse than sieve in high dimensions, but it requires much less space, and thus it is more suitable for massive parallelization with small memory. The basic procedure is as follows:

(i) We first execute ENUM on the sub-lattice $\mathcal{L}(\{\mathbf{b}_1, \ldots, \mathbf{b}_{k-1}\})$.

(ii) Then we find a short lattice vector $\mathbf{v}$ by performing child-ENUM over the projected lattice $\pi_k(L)$ such that $\mathbf{v} \in D_{k,\tau}$. If we finish searching all nodes of the enumeration tree of $\pi_k(L)$, we output the shortest vector found and finish the Sub-ENUM.

(iii) We then run root-ENUM on the sub-lattice $\mathcal{L}(\{\mathbf{b}_1, \ldots, \mathbf{b}_{k-1}, \mathbf{v}\})$. If $\pi_k(\mathbf{s}) = \pi_k(\mathbf{v})$ is satisfied, then we can obtain the shortest vector $\mathbf{s} \in L$ by root-ENUM, else we back to step (ii).

Chapter 4

CMAP-LAP: Framework for solving lattice problems with massively parallelization

In this chapter, we propose a framework CMAP-LAP that implements a new parallel, multi-algorithm scheme for lattice problems based on a supervisor-worker parallel system. Our framework allows several different single-rank or multi-rank solvers to work cooperatively while efficiently sharing information with other solvers, even in a large-scale computational environment. This framework is based on the Generalized UG framework (described detail in Section 4.2), but the information managed by the supervisor and the data structure communicated are customized for the lattice problem and it was created from scratch. In Section 4.1, we show the motivation and overall design of our framework and expanded data pool for lattice problems. In addition, we describe some implementation techniques, checkpoint-and-restart functionally, hybrid parallelization, non-blocking communication. In Section 4.3, we show the performance of CMAP-LAP, such as sharing efficiency, scalability and checkpoint-and-restart functionally.

4.1 Design of framework

It is essential for a practical solver to utilize the multiple lattice algorithms introduced in Chapter 3. Most of the existing solvers discussed in Section 1.3 rely on either the combination of lattice reduction and sieve or the combination of lattice reduction and ENUM. These algorithms are inter-dependent and executed sequentially. In contrast, CMAP-LAP is built on a new multi-algorithm paradigm in which multiple lattice algorithms are executed cooperatively and yet asynchronously in parallel. The key idea is that each lattice algorithm described in Chapter 3 can be considered a *sampler* of short lattice vectors. Furthermore, each algorithm benefits from the knowledge of short vectors; for example, the enumeration tree of ENUM shrinks according to the upper bound R of the shortest norm. Using different algorithms and randomly transformed bases, we can increase the number of samplers,

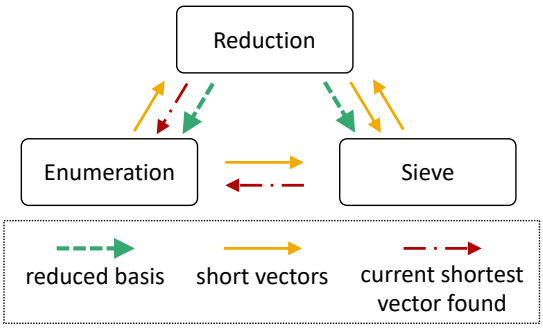

FIGURE 4.1: Interaction among SVP algorithms

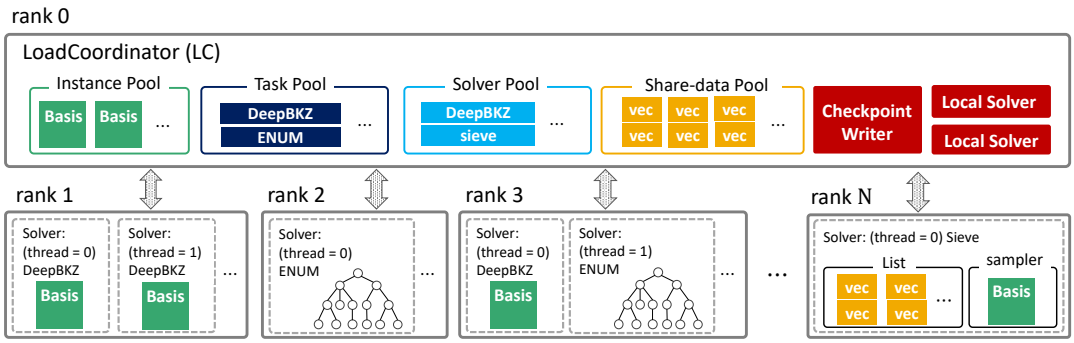

FIGURE 4.2: System overview of CMAP-LAP for lattice problems

which mutually boosts the sampling performance by sharing the information of short vectors found (see Figure 4.1). To realize the novel multi-algorithm paradigm, CMAP-LAP was developed entirely from scratch utilizing the full power of the Generalized UG, which is a generic high-level task parallelization framework.

4.1.1 Architecture

We describe the architecture of CMAP-LAP. The Generalized UG consists of a controller process, *LoadCoordinator* (`LC`), and multiple *Solver*s. Each `Solver` communicates with `LC` asynchronously. This system is suitable for multiple processes that run different algorithms and share information as needed. CMAP-LAP adopts the Supervisor-Worker load coordination paradigm (see [Ral+18]), where `LC` is supervisor and `Solvers` are workers. The main difference to the typical master-worker paradigm is that the supervisor's task is limited and workers act more independently by exchanging small messages with supervisors as needed, avoiding unnecessary overhead to manage workers. The `LC` has the following data pools: (i) Instance Pool, (ii) Solver Pool, (iii) Task Pool, and (iv) Share-Data Pool. (See Figure 4.2). The `LC` creates particular purpose local threads as needed: (i) Checkpoint Writer thread (ii) Local solver threads.

Each `Solver` carries a *Task*, which is a triple of:

- *Instance* is the data that represents the problem to solve, which in the case of SVP is a lattice basis, and in the case of CVP is a lattice basis and a target vector.
- *Parameters* describe the type of algorithm and the parameters of the algorithm—for example, an ENUM algorithm with a pruning strategy from *Parameters*.
- *Status* represents the algorithm's progress, e.g., for the depth-first search of the enumeration algorithm, it is the node currently being searched.

Given a lattice problem, each `Solver` is created in one core and assigned a *Task* by `LC`. The basic flow of CMAP-LAP is as follows (see Figure 4.3):

(1) `LC` stores given *Instance* in the instance pool.

(2) `LC` pops an *Instance* from the instance pool, sets *Parameters* for *Instance*, and initializes *Status*. The created *Task* = (*Instance*, *Parameters*, *Status*) is stored in the task pool.

(3) If there exists an idle `Solver`, `LC` pops a *Task* in the task pool and sends it to the idle `Solver`, and stores it to the solver pool.

(4) Each `Solver` takes the algorithm and its input from the received *Task*, and occasionally shares information to `LC`, such as *Instance*, Data, *Status*. The information sent depends on the algorithm, as shown in Figure 4.1. `LC` stores the information in the pool according to this type. In addition, `Solver` sends its *Status* to `LC`, and `LC` updates *Task* in the solver pool for the checkpoints.

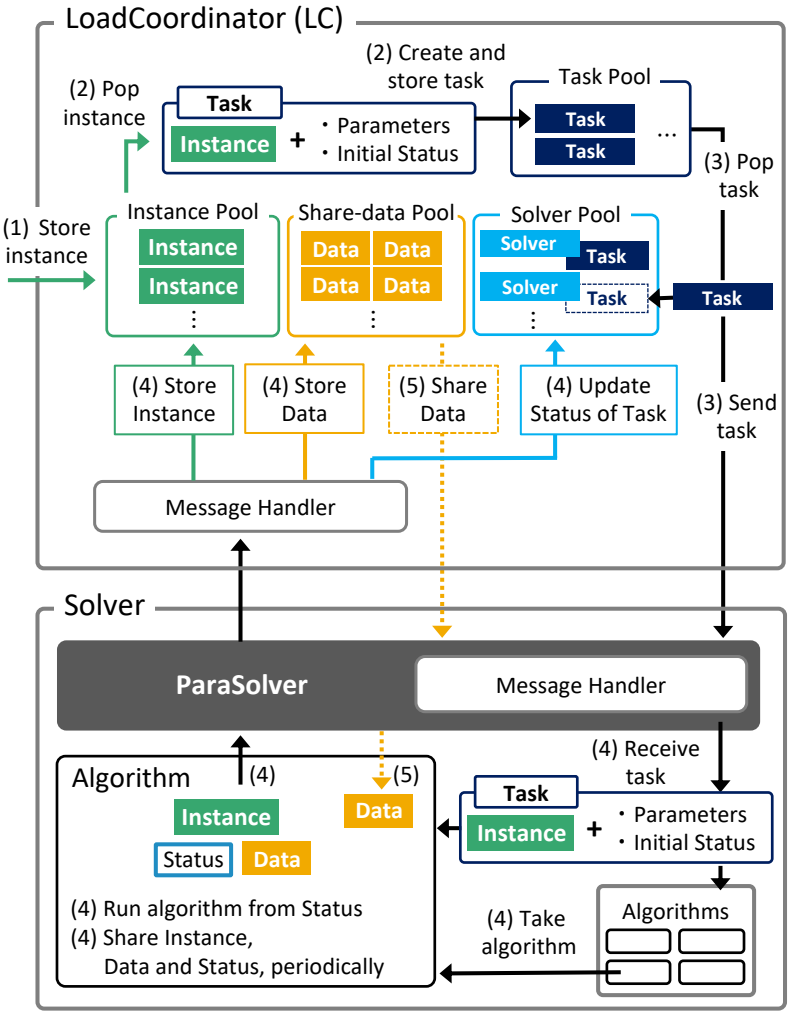

FIGURE 4.3: Execution flow of CMAP-LAP

(5) Information in the share-data pool is occasionally retrieved from `LC`, and shared among `Solvers`. Each `Solver` updates its *Parameters* according to the shared information. See Chapter 3 for how the shared information is utilized by each algorithm run by the `Solver`.

(6) When a `Solver` finishes the assigned *Task*, it sends its final *Status* to `LC` and becomes idle.

`LC` always checks for messages from `Solver`. Messages received by the `LC` are processed through the message handler according to the type of message. As described above, `Solver` only communicates with `LC`, and `Solver` does not share information with other `Solvers` directly. This communication via the share-data pool is an effective solution for massive parallelization to achieve (i) the reduction in the number of communication paths, (ii) the management of the total amount of communication, (iii) the control over the memory usage, and (iv) I/O for checkpoint and progress takes place solely within `LC`.

The detail of the components of CMAP-LAP is given as follows.

Instance Pool Instance pool stores instances of the problem together with their priorities. For example, bases transformed by unimodular matrices give the same lattice and represent different instances of the same lattice problem. The instance pool is initialized with the single basis provided a lattice basis that specifies the lattice problem. `LC` stores bases sent from `Solvers`, which run a lattice reduction algorithm. In the case of SVP, the priority can be computed by the estimated total number of nodes in the enumeration tree described in Section 3.1 such that the shortest vector will be found more efficiently with an instance of

higher priority. LC pops an instance with the highest priority from the instance pool and creates a *Task* from it.

Task Pool Task pool stores *Task*s, which are triples of (*Instance*, *Parameters*, *Status*). It manages the *Task*s waiting to be executed. LC assigns the *Task* with the highest priority to a Solver. In this way, the *Task*s, which would lead to better solutions quickly, are prioritized. Multiple *Task*s may be generated from a single instance using different algorithms and parameters.

Solver Pool Solver pool stores information of the running Solvers. Each Solver is managed by (Solver Id, *Task*). The *Status* of *Task* is periodically updated by the *Status* message sent from Solver. This mechanism allows LC to grasp the status of all Solvers. When Solver finishes the assigned *Task*, it is registered as idle. In addition, when LC wants to assign a new *Task* of high priority immediately, LC chooses a running Solver to interrupt the current *Task*.

The number of active Solvers that runs on a single machine node is determined by LC according to the computational cost of *Task*. For example, sieve algorithms have a large memory footprint to maintain a large number of lattice vectors; a single Solver becomes active and runs on a single machine node. Meanwhile, ENUM and reduction algorithms use little memory, and the same number of Solvers as that of the cores run on a single node.

Share-Data Pool Share-data pool stores information that is shared across multiple Solvers. In the case of CMAP-LAP, a typical type of information sent from Solvers is a lattice vector of the small norm. The size of the message is equal to the product of the dimension (e.g., 130) and the size of the scalar (e.g., long integer). LC checks if the sent vector is already in the pool. If it is not in the pool, an entry (Data, Sent-Solvers, priority) is created and added to the pool, where Data is the sent vector. Sent-Solvers is a set that records the Solver Ids to which Data has been sent. The priority is computed by its norm. When the pool size gets bigger, LC decides which entries remain stored in the pool according to their priorities. At an interval, LC selects an entry according to the priority and pushes it to those Solvers whose Solver Ids are not in Sent-Solvers and adds their Solver Id to Sent-Solvers. In this way, information is shared among all Solvers efficiently while controlling the total amount of communication. The interval at which Solvers and LC push information can be tuned depending on the configuration of the machine. There is no danger of locking regarding the order of messages in our scheme.

The share-data pool is the most memory-consuming part of the LC. The size of the share-data pool increases over time, and the limit of the pool size must be set appropriately according to the available memory. In particular, the size of the Sent-Solvers is dominant and should be estimated carefully in case of massive parallelization. Moreover, the cost of Data retrieval increases when the pool size and the number of Solvers are large. In this case, the limit of the pool size and the frequency of data sharing are suppressed.

Fully Checkpoint Functionality with Checkpoint Writer thread One of the most powerful features of CMAP-LAP is the checkpoint mechanism for storing high-level information of the whole system. Lattice problems are hard and often require millions of core hours. Thus, it is critical to have the functionality to record the progress and resume after interruption. Our checkpoint functionality is carefully designed to store high-level, platform-independent information to enable restart even on different platforms.

When a checkpoint is requested, the data in the pools in LC are serialized and stored in checkpoint files using zlib [DG96], a portable compression library. At the time of restart, CMAP-LAP reads the checkpoint files to restore pools. The task pool contains *Task*s, including the progress information *Status*, which can be assigned to Solvers to resume. When the checkpoint files are loaded in a different environment from the one that has saved them, the number of cores and the available memory may differ. In this case, LC distributes the *Task*s in the task pool to Solvers as much as possible, leaving the other *Task*s in the task pool. At the same time, LC creates new *Task*s when a large number of Solvers are available.

The technically important point is that the message processing from Solvers to LC is blocked when LC writes checkpoint files. With many MPI packages, this is problematic

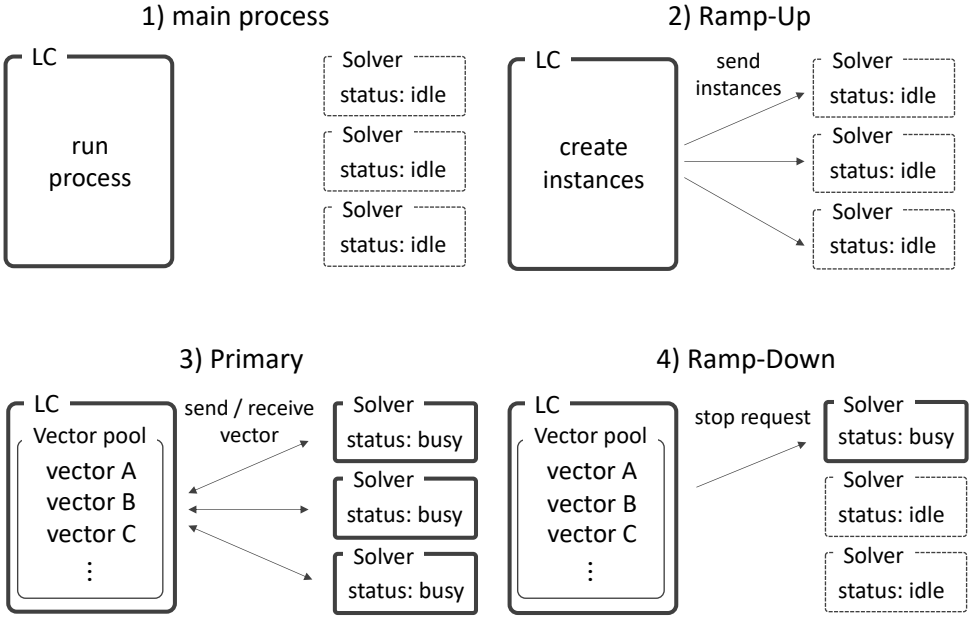

FIGURE 4.4: Basic phases of the parallel dispatch

because the size of the queue of MPI messages waiting to be received becomes large and eventually leads to an error when the upper limit is reached. This problem becomes more pronounced as the scale of execution increases. To avoid this problem, `LC` temporarily creates a copy of the pools on memory, and a dedicated thread in `LC`, called *Checkpoint Writer*, is created to write the copy in the checkpoint files. Using the Checkpoint Writer thread has significantly reduced the block time for checkpoints and enabled CMAP-LAP to run stably on large-scale platforms.

Local Solver threads Some solvers can be created as a thread in `LC`. These *Local Solver*s work on lightweight tasks requiring access to the entire pools. For example, Local Solvers list the projected vectors in the share-data pool, which are found by `Solvers` performing sub-ENUM and sub-sieve introduced in Section 3.4. Because Local Solvers have access to the share-data pool without communication, the total amount of communication is reduced in this way.

4.1.2 Parallel dispatch

Here, we describe *parallel dispatch*, which is a comprehensive execution flow in CMAP-LAP. The parallel dispatch executes one parallel computing of one (sub-) problem as one cycle. This parallel dispatch is essential to parallel for solving exact SVP in BKZ algorithm. The parallel dispatch consists of four execution phases: *main process*, *Ramp-Up*, *Primary*, and *Ramp-Down*, as shown in Figure 4.4. With these four phases as one cycle, parallel dispatch executes the cycle multiple times.

Main process phase Herein, only the `LC` runs the process, and all `Solvers` are idle. `LC` obtains the results of the parallel computation, prepares for the next parallel computation, and performs other operations.

Ramp-Up phase (pseudocode is Algorithm 6) This is the period from when all `Solvers` are idle until when all `Solvers` start processing after receiving the instance. `LC` creates an instance and sends it to the `Solvers` in turn. Therefore, some `Solvers` are delayed in receiving instances. We call the waiting time until these `Solvers` start processing *start idle time*.

Algorithm 6 Ramp-Up Phase

```
1: while LC should create and assign Task do
2:     LC send Task to an idle Solver;
3: end while
```

Primary phase (pseudocode is Algorithm 7) All Solvers are processing the given instance. During this and the Ramp-Down phase, the Solvers send or receive vectors objects to and from the LC asynchronously. It allows all Solvers to share information through LC. LC has a priority queue called the *vector pool*, which stores vector objects. When a Solver sends the vector objects to the LC, the LC stores them in the vector pool if necessary. Conversely, if a Solver sends a receive request to the LC, the LC sends the appropriate vector objects from the vector pool to the Solver. Each Solver can send and receive the vector objects at its own convenient timing. The vector pool is managed with customized priorities, and when the pool is full, the vectors with lower priorities are removed. Because the LC receives send and receive requests from multiple Solvers, a time lag occurs during sending and receiving. We call it *wait idle time*.

Algorithm 7 Primary Phase

```
1: while True do
2:     while LC should create and assign Task do
3:         LC sends Task to an idle Solver;
4:     end while
5:     LC checks to have received any messages from Solvers;
6:                                 ▷ LC calls handlers according to type of received messages;
7:     if LC should create checkpoint then
8:         LC creates checkpoint;
9:     end if
10:    while LC should create and assign Task for LocalSolver do
11:        LC sends Task for LocalSolver to an idle LocalSolver;
12:    end while
13:    if There is no active Solver then
14:        break;
15:    end if
16: end while
```

Ramp-Down phase (pseudocode is Algorithm 8) This is the period when at least one Solver is in an idle state. Information is shared between busy Solvers through LC. The end time of a Solver depends on the instance given, or the vector received from the LC. If there are many Solvers, the end time variance generally becomes large. Therefore, LC has a function that can send a stop request to a Solver. When a Solver receives a stop request, it ends the process immediately.

Algorithm 8 Ramp-Down Phase

```
1: LC send TerminateTag to all active Solvers;
2: while There are active Solvers do
3:     LC wait TerminateTag from Solvers;
4: end while
```

Solver process (pseudocode is Algorithm 9) Finally, we show a brief pseudo-code for the solver process. Idle Solvers wait for a *Task*from LC, and when Solver receives the *Task* from LC, Solver executes the algorithm according to the received *Task*, information of the algorithm and its arguments. By passing the ParaSolver communication API of CMAP-LAP to the algorithm, we can share the information of lattice basis and lattice vectors and receive

Algorithm 9 `Solver` Process

```
1: while True do
2:     Solver wait a message from LC;
3:     if Solver receives new Task then
4:         break;
5:     else if Solver receives TagTerminate then
6:         Solver send statistics data to LC, and send TagTerminate;
7:         return;
8:     end if
9:     Solver run algorithm according to received Task;
10: end while
```

the termination notification through the communication API. When the `Solver` receives a TagTerminate from `LC`, it sends the statistics of the executed *Task*s to `LC` and send-backs the TagTerminate to `LC`. By keeping a count of the number of TagTerminates `LC` has received, `LC` can terminate after all `Solver` processes have been finished. This is the safest termination.

4.2 Implementation

Generalized UG: A framework to construct CMAP-LAP We have built a solid backbone to manage hundreds of thousands of processes running heterogeneous algorithms in parallel by specializing *Generalized UG framework* (UG version 1.0 RC). The Generalized UG framework is extending the well-recognized Ubiquity Generator (UG) framework [Ug] for Branch-and-Bound (B&B) algorithms. UG framework is a generic software framework to parallelize an existing state-of-the-art B&B based solver, which is referred to as the *base solver*, from "outside". UG is composed of a collection of base C++ classes, which define customizable interfaces to base solvers and translate solutions and subproblems into a solver independent form. Additionally, a base class defines interfaces for different message-passing protocols corresponding to the parallelization library used. UG has been developed primarily in concert with a state-of-the-art mixed integer programming solver called SCIP [Sci]. As such, ParaSCIP [Shi+11], and FiberSCIP [Shi+18a], which run on a distributed computing environment and shared memory computing environment, respectively, are the most mature. Notably, the distributed memory and shared memory solvers execute the same algorithm in general for the instantiations. UG has been successfully utilized for mixed-integer linear programming problems [Shi+18b; Shi+16; SBH18], Steiner tree problems [Gam+17; SRK19; SRG19; RSK21], and quadratic assignment problems [Fuj+21] on supercomputers.

UG has shown flexibility and scalability for solving optimization problems. The ability of UG motivated the development of a parallel solver using a non-B&B based solver. Generalized UG has been developed to enable parallelization of such a non-B&B based solver. Generalized UG consists of several abstract classes which can be customized according to the target problem. This customization flexibility is suitable for the realization of various parallel strategies.

Extendability There are many lattice problem solvers, including the state-of-the-art sieve solver G6K, which is available as open-source software. CMAP-LAP's flexible and highly modular design allows solvers to be incorporated as a part of the system. For the ease of incorporation, an interface class *ParaSolver* is provided, with which existing solvers can be turned into `Solvers` with minimum effort. Each `Solver` has a ParaSolver object that takes care of all the communication, and existing solvers only have to receive input data and send the results via ParaSolver's API (see bottom of Figure 4.3). The solvers are not limited to single-rank applications. The UG has a feature to parallelize multi-rank applications. See [Mun+19] as an example.

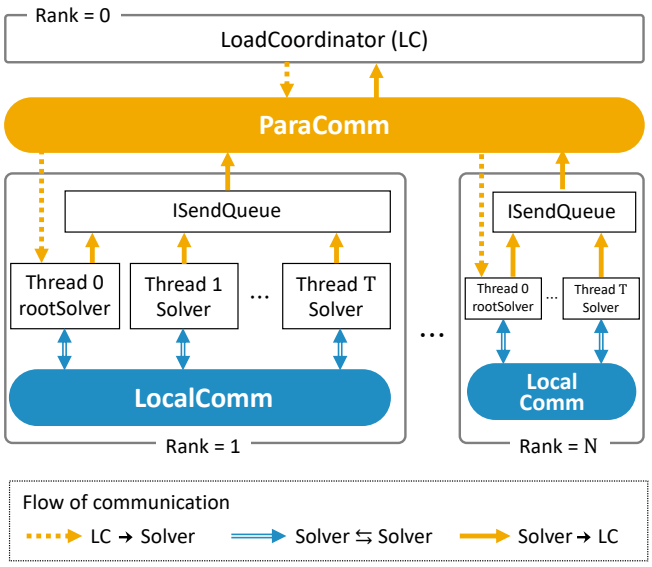

FIGURE 4.5: Communicators between and within MPI processes: ParaComm and LocalComm

Hybrid parallelization CMAP-LAP uses hybrid parallelization that combines MPI communication with C++11 thread communication. `LC` and Solver have two kinds of communicators: one is *ParaComm*, which wraps MPI communication functions, and the other is *LocalComm*, which wraps C++11 communication functions. ParaComm is used for inter-process communication, and LocalComm is used for inter-thread communication within a process (see Figure 4.5). Because all `Solvers` know the MPI rank of `LC`, Solvers send messages directly to `LC` using ParaComm and ISendQueue, which is described in the following section. In contrast, when `LC` sends a message to `Solver`, `LC` first sends a message via ParaComm to the MPI rank where the `Solver` resides. The solver with 0 thread-Id receives the message; we call this the *rootSolver*. Then, the rootSolver sends the message to the `Solver` using LocalComm. Therefore, the rootSolver receives more messages than the other `Solvers`, the received messages must be checked frequently, even during the execution of the algorithm. However, the idle time for message processing can be reduced by using non-blocking communication, as described below.

MPI_ISend communication Because `LC` receives messages from all busy `Solvers`, the `LC`'s load is the highest of all the processes in the case of large-scale computation. In addition, depending on the type of messages received, processing such as inserting Data into the share-data pool occurs in `LC`. This blocks the `LC` message processing and delays the receiving of the messages. Note that the load coordination paradigm used in CMAP-LAP is Supervisor-Worker [Ral+18] and then small message communications are performed between `LC` and `Solvers` for load balancing. Although the frequency for the small message communications can be controlled by run-time parameters, they are crucial in large-scale computations such as over $100,000$ `Solvers` used. Therefore, in CMAP-LAP, to reduce the idle time of communication in `Solver`, we send all messages from `Solver` to `LC` by using MPI_ISend, the non-blocking communication. This leads `Solver` to resume the algorithm without waiting for the check that `LC` receives the message. To prevent the objects deleted before they are sent, we copy the objects sent by MPI_ISend to a queue called *ISendQueue* in the memory of that process. We remove them from ISendQueue as soon as the transmission is confirmed by MPI_Test (see Figure 4.6). By examining the size of each ISendQueue, we can determine the number of unreceived messages of `LC`. Therefore, we set an upper limit on the size of ISendQueue and do not send messages exceeding the limit, thereby preventing many messages from accumulating in `LC`.

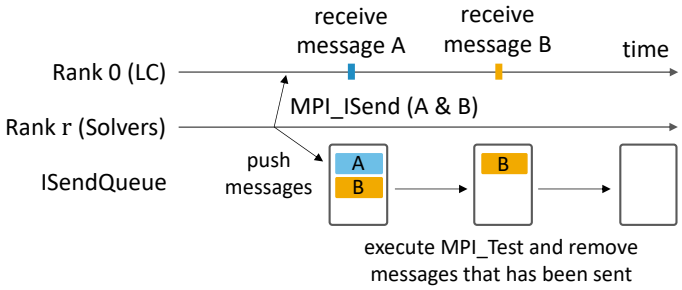

FIGURE 4.6: MPI_ISend Communication between `Solver` and `LC`

4.3 Performances of Framework with Testing Configure

In this section, we evaluate the performance of CMAP-LAP with the SVP challenge. The computing platform used in the following numerical experiments includes the Lisa and Emmy at Zuse Institute Berlin, and ITO at Kyushu University. These specifications are summarized in Table 4.1.

TABLE 4.1: Computing platforms used

Machine	Memory / node	CPU	CPU frequency	# nodes	# cores
Lisa	384 GB	Xeon Platinum 9242	2.30 GHz	1,080	103,680 (96 × 1,080)
Emmy	384 GB	Xeon Platinum 9242	2.30 GHz	128	12,288 (96 × 128)
ITO	192 GB	Xeon Gold 6154	3.00 GHz	128	4,608 (36 × 128)
CAL A	256 GB	Xeon E5-2640 v3	2.60 GHz	4	64 (16 × 4)
CAL B	256 GB	Xeon E5-2650 v3	2.30 GHz	4	80 (20 × 4)

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.

Setting up CMAP-LAP for testing to solve SVP We briefly describe the overall behavior of CMAP-LAP for solving SVP. Recall that an SVP is specified by a lattice basis matrix. At the beginning of the execution, the `LC` reads the basis matrix from a file and stores it in the instance pool. `LC` creates a Local Solver to transform the basis with random unimodular matrices and stores the resulting bases in the instance pool. Then, `LC` generates DeepBKZ *Task*s for the bases in the instance pool. The reduced bases are sent from `Solvers` performing DeepBKZ *Task*s to `LC`, and `LC` stores them in the instance pool. `LC` also generates ENUM and sieve *Task*s using the bases in the instance pool. Short lattice vectors are occasionally sent from `Solvers` to `LC`, which are inserted into the share-data pool. At regular intervals, `Solvers` request `LC` to send short vectors from the share-data pool. DeepBKZ *Task*s insert the received short vectors into the basis, sieve *Task*s use the received short vectors as sampling seeds, while ENUM adjusts the search radius according to the norm of the shortest vector ever found. Some of the contents of the pools in `LC` are written to checkpoint files at regular intervals: vectors in the data-share pool, basis matrices in the instance pool, and *Task*s in the solver pool. The *Task* mainly contains the basis matrix and the vector and parameters needed to run the algorithm. When restarting, as described in Section 4.1.1, there are few processes other than reading checkpoint files. We calculate the communication interval and the number of vectors shared from the number of cores, and the maximum MPI buffer size to relax the communication delay.

Since computing the exact norm of a shortest vector of a given lattice is as hard as computing a shortest vector, we evaluate the progress of solving an SVP instance by the approximation factor defined in Chapter 3. A smaller value of the approximation factor indicates a

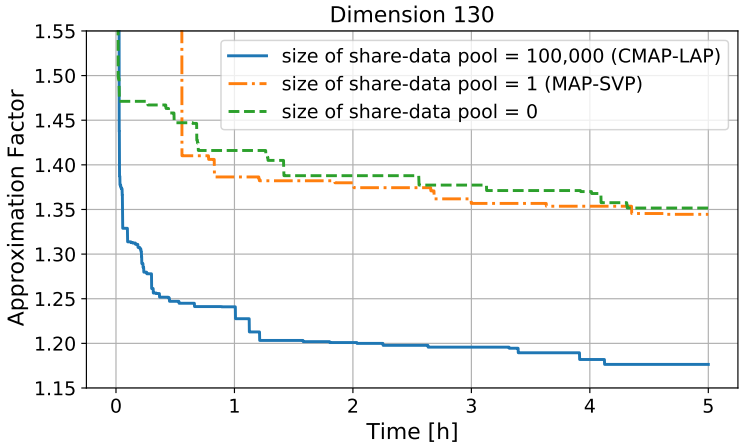

FIGURE 4.7: Transition of the approximation factors for different share-data pool sizes; execution were done on the CAL A and CAL B with 144 cores. The solid blue lines in Figure 4.7, 4.9 and 4.11 represent the same experimental result.

better (temporary) solution. With the Gaussian Heuristics, the approximation factor should be about 1.0 for a good candidate of a shortest vector. From a cryptanalysis viewpoint, an approximate factor of 1.05 is often set as a goal as in the SVP challenge. The numbers of lattice vectors having smaller approximation factors decrease quickly; for example, in dimension $n = 130$, the ratio of the numbers of lattice vectors having approximation factors 1.20 and 1.30 is approximately $(1.20^n/1.30^n) \approx 3.03 \times 10^{-5}$. In other words, it is 33,000 times harder to reach an approximate factor of 1.20 than of 1.30. It becomes increasingly harder to find lattice vectors with smaller approximate factors; for example, the ratio of the numbers of lattice vectors having approximation factors 1.10 and 1.20 is approximately $(1.10^n/1.20^n) \approx 1.22 \times 10^{-5}$.

4.3.1 Information sharing

We evaluate the effect of our novel information-sharing scheme and the parallelization with the lattice reduction algorithm. We performed experiments running DeepBKZ with $\beta = 30$ for five instances of the SVP challenge of dimension 130 with seeds from 0 to 4. We executed all computations on the CAL A and CAL B with 144 cores.

We show the efficiency of the information sharing with CMAP-LAP. In CMAP-LAP, `Solvers` share multiple short lattice vectors via the share-data pool in `LC`. The amount of information shared among `Solvers` can be controlled by the size of the share-data pool. Figure 4.7 compares the transition of the approximation factor (averaged over 5 instances) overtime with the size of the share-data pool 0, 1, and 100,000. When the size of the share-data pool is set to zero, no information is shared and all the `Solvers` are executed independently. When the size of the share-data pool is set to 1, only the current shortest lattice vector (the current solution) is shared among `Solvers`. This is equivalent to the sharing scheme of MAP-SVP. We observe that the approximation factor is drastically reduced when the size of the share-data pool is set to 100,000. This shows the effectiveness of our data sharing scheme.

4.3.2 Coordination of heterogeneous algorithms

We show the effectiveness of CMAP-LAP's multi-algorithm paradigm, in which heterogeneous lattice algorithms are executed concurrently in coordination. In this experiment, we fix the number of `Solvers` assigned to each *Task*, that is, DeepBKZ, sub-ENUM, and GaussSieve. Each `Solver` is assigned the the same type *Task* when it completes the current *Task*. Figure 4.8 and 4.9 shows the results for a 110- and 130-dimensional SVP with four different configurations of the *Task* assignment, respectively. We ran the experiment on the CAL A and CAL B with 144 cores for an hour or five hours, and the 1 core was assigned to `LC`, and

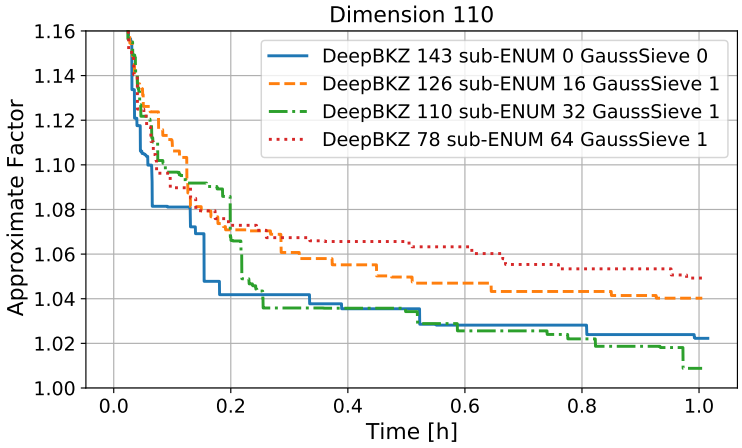

FIGURE 4.8: Same as Figure 4.7, but dimension is 110 and different allotment of algorithms; execution were done on the CAL A and CAL B with 144 cores.

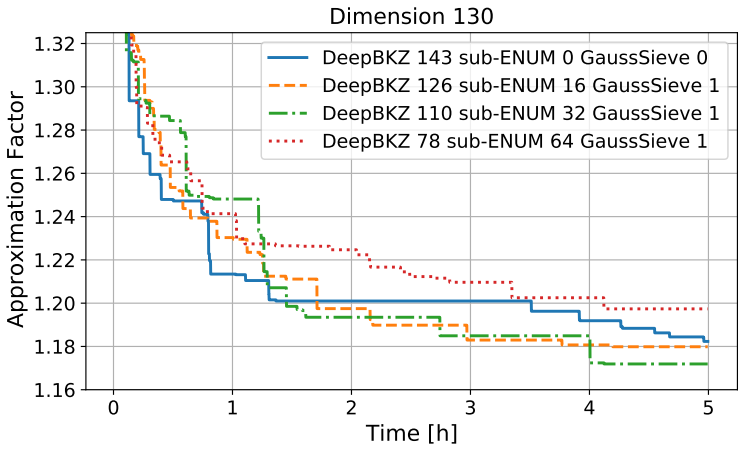

FIGURE 4.9: Same as Figure 4.7, but for different allotment of algorithms; execution were done on the CAL A and CAL B with 144 cores.

the other 143 cores were assigned to three types of *Task*s. We set the size of the share-data pool to be infinity. The best result was obtained with the combination of (DeepBKZ, sub-ENUM, GaussSieve) = (110, 32, 1). To investigate the reason, we examine the distribution of vector norms in the share-data pool for two configurations of 130-dimensional experiments (see Figure 4.10). The total number of vectors shared through the share-data pool for (DeepBKZ, sub-ENUM, GaussSieve) = (143, 0, 0) was 36,055, and that for (DeepBKZ, sub-ENUM, GaussSieve) = (110, 32, 1) was 101,952. In both configurations, shorter vectors were found by DeepBKZ `Solver`. However, a large number of relatively short vectors found by sub-ENUM and GaussSieve helped DeepBKZ find shorter vectors.

4.3.3 Scalability

To see the scalability of CMAP-LAP, we experimented with the same 130-dimensional SVP instances as in Section 4.3.1 on Lisa using 2,976, 6,048, 12,192, 24,480, and 49,056 `Solvers` with DeepBKZ ($\beta = 30$). We measured the average number of the main iterations (called the *tour*) performed by each `Solver` within six hours. The number of tours provides a good estimation of the progress of the DeepBKZ algorithm. As we observe from Table 4.2, the average number of tours stay almost constant when the number of `Solvers` increases. Therefore, even if the number of `Solvers` becomes large-scale, there is no significant change in the performance of each `Solver`.

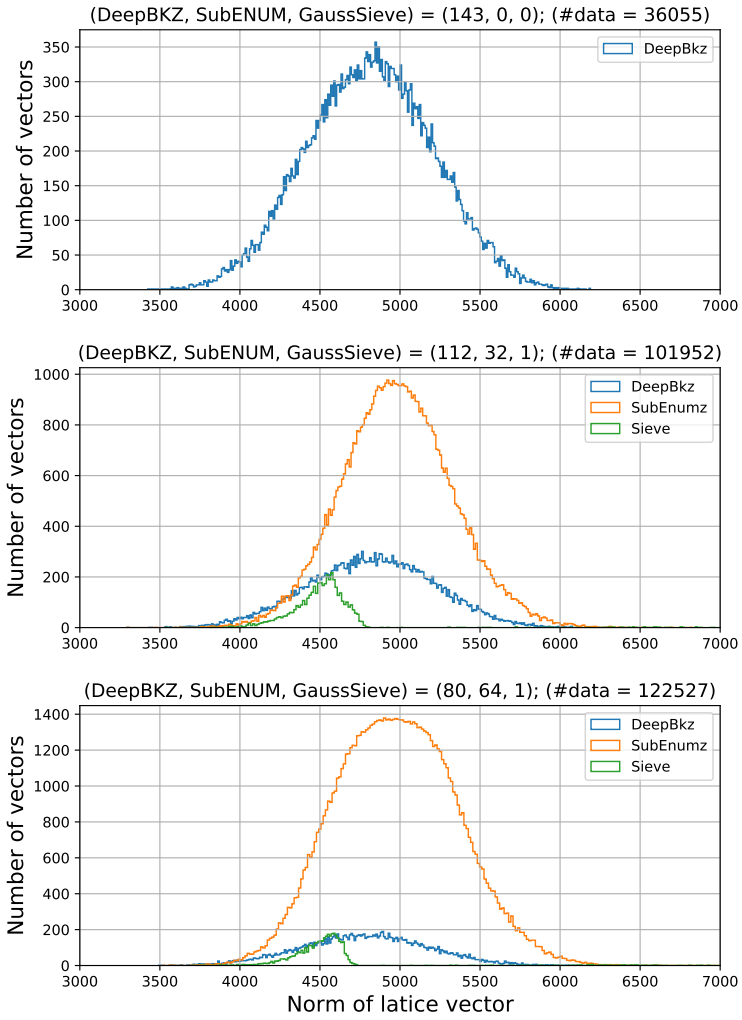

FIGURE 4.10: Distribution of the norm of vectors in the share-data pool.

In addition, we evaluated the effect of parallelization on the transition of the approximation factor (see Figure 4.11). We experimented with the same SVP instances as in Section 4.3.1 using different numbers of `Solvers`. The size of the share-data pool was set to 100,000. We used the CAL A and CAL B with 144 cores and ITO with 2,304 cores for this experiment. The best (minimum) approximation factor obtained within 5 hours with 143 `Solvers` was 1.176 and 1.117 with 2,303 `Solvers`. In terms of Gaussian Heuristics, the latter is considered to be $1.176^{130}/1.117^{130} \approx 800$ times better. It took 14,844 seconds to reach the approximation factor of 1.176 with 143 `Solvers` while it took 2,965 seconds with 2,303 `Solvers`, which is a speed-up by a factor of 5.0 compared with 143 `Solvers`. Similarly, the time for the approximation factor to fall below 1.2 was 7,319 seconds with 143 `Solvers` and 1,360 seconds with 2,303 `Solvers`, which is a speed-up by a factor of 5.3.

4.3.4 Stability with massive parallelization

We show the results of a long-time execution of CMAP-LAP.

TABLE 4.2: Iterations of DeepBKZ of each `Solvers` for 130-dimensional SVP

# of `Solvers`	2,976	6,048	12,192	24,480	49,056
averaged # of iterations	45.86	43.32	43.08	40.10	57.07

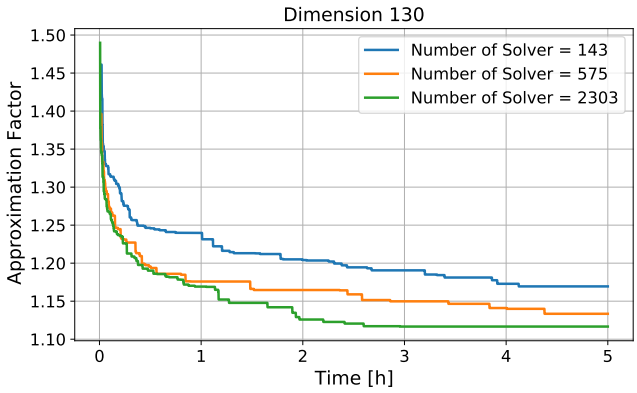

FIGURE 4.11: Same as Figure 4.7, but for different number of `Solvers`; execution were done on the CAL A and CAL B with 144 cores, and ITO with 2,304 cores.

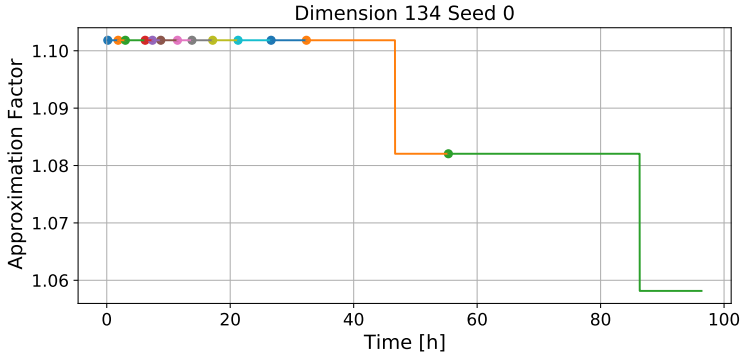

FIGURE 4.12: Transition of the approximation factor of a 134-dimensional SVP for long-time execution on the Lisa with 103,680 cores. Each dot represents the beginning of restart from checkpoint.

Figure 4.12 shows the result of multiple executions of a 134 dimensional SVP instance. We ran the experiment 13 times using our checkpoint-and-restart functionality on the Lisa supercomputer with 103,680 cores. The first few executions were performed for short periods to test the checkpoint functionality. During the test, we observed occasional aborts due to an excessive number of MPI messages waiting to be received by the `LC`. As a workaround, the Checkpoint Writer (described in Section 4.1.1) was developed, and the upper limit of the size of ISendQueue was set based on the number of messages the `Solver` sends to the `LC` (described in Section 4.2). This has improved the stability and enabled a longer execution time. We have tested up to 42 hours of continuous execution. Together with checkpoint and restart, the approximation factor was improved over time.

Figure 4.13 shows the result of multiple executions of a 130 dimensional SVP. This time, we tested a restart from a checkpoint created on a different environment. The first 14 executions were performed on the Emmy with 12,288 cores and the last 1 execution was restarted on the Lisa with 103,680 cores. Although the number of cores used in the Lisa is 8.44 times more than that of the Emmy, the execution was carried over by the checkpoint functionality without any problem. The *Task*s running on the Emmy when the checkpoint was created were executed on the Lisa immediately after the restart, and new *Task*s were generated from the instance pool and assigned to extra `Solvers` available on the Lisa. It should be noted that the approximation factor was improved in the last execution after the final restart (see the purple segment in Figure 4.13).

The interval of the creation of checkpoint files were set to an hour. It took an average of 1,531.75 seconds per checkpoint for the Checkpoint Writer to compress and write the pool's information in files, whose size was approximately 7.09 GB on memory. In contrast, it took only an average of 2.77 seconds for `LC` to copy the pools for the Checkpoint Writer. In this

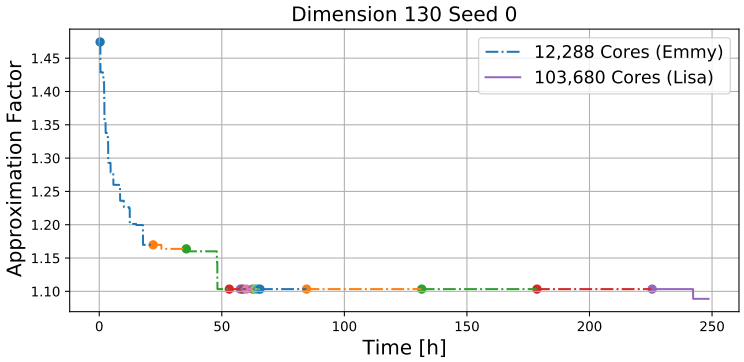

FIGURE 4.13: Transition of the approximation factor of a 130-dimensional SVP for long-time execution on the Emmy with 12,280 cores and Lisa with 103,680 cores.

manner, the blocking time of LC's message processing was greatly improved by the Checkpoint Writer. The averaged time required to read a checkpoint is only 64.75 seconds, and since the checkpoint contains *Task* information, execution can be restarted without prepossessing.

Chapter 5

CMAP-DeepBKZ: Software for DeepBKZ with massively parallelization

In this Chapter, we propose and implement a parallel strategy specialized for lattice basis reduction based on the CMAP-LAP framework, one of the algorithms for SVP. By taking advantage of the information sharing feature of CMAP-LAP, the lattice basis reductions are executed concurrently in multiple processes with sharing a part of the basis among all worker processes. There is a trade-off between this information sharing and the randomness among the basis, and it is important to balance them. In order to obtain the full benefit of parallelization, it is essential that, at least, different computations are performed by multiple processes. However, to the best of our knowledge, no metric has been proposed to quantify the diversity of basis sets. Therefore, we propose a metric to quantify the diversity of a set of bases and verify its validity. Using this metric, we experimentally confirm that the independence of parallel computation will be kept even after information sharing. Finally, we evaluate the performance of the proposed software in detail by experiments using up to about 100,000 cores. As a result, we succeeded in improving the quality of the output basis and updating the SVP challenge record of up to 128 dimensions.

5.1 Parallel strategy

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 in every solver through CMAP-LAP featuress.

5.1.1 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 $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_m)$ of L such that $\|\mathbf{b}_j^*\| = \lambda_1\left(L_{[j,k]}\right)$ for all indices j with $k = \min(j+\beta-1, m)$, by calling an SVP oracle (e.g., an enumeration algorithm in Algorithm 5) on the projected lattice $L_{[j,k]} = \mathcal{L}(\mathbf{B}_{[j,k]})$ cyclically for $j = 1, 2, \ldots, m-1$. During DeepBKZ reduction, the Gram-Schmidt norms $\left(\|\mathbf{b}_1^*\|, \ldots, \|\mathbf{b}_m^*\|\right)$ decrease monotonically in lexicographic order. As β increases, the quality of an output basis improves in both theory and practice. Similar to BKZ, when $\beta = m$, 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 $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_{m_1})$

and $\mathbf{C} = (\mathbf{c}_1, \ldots, \mathbf{c}_{m_2})$ of a lattice, we define an order as

$$\begin{aligned} &\mathbf{B} < \mathbf{C} \iff \\ &\exists j \leq \min\{m_1, m_2\} \text{ s.t. } (\|\mathbf{b}_i^*\| = \|\mathbf{c}_i^*\|) \wedge (\|\mathbf{b}_j^*\| < \|\mathbf{c}_j^*\|) \text{ for all } i < j. \end{aligned} \tag{5.1}$$

(Cf., for a parameter $0 < \Theta < 1$, the authors in [TKH18] considered $\sum_{i=1}^{m} \Theta^i \|\mathbf{b}_i^*\|$ as the score of $\mathbf{B}$ for their random sampling reduction algorithm.)

5.1.2 Strategy of parallel sharing in DeepBKZ

Our aim of parallelization is to efficiently find a small lattice basis in the lexicographic order (5.1) 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 (5.1). 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 (5.1) for all bases of the solver $\mathbf{B}$ of L. For a global basis $\mathbf{S} = (\mathbf{s}_1, \ldots, \mathbf{s}_m)$ of L, we also call its sub-basis of the form $\mathbf{S}_k = (\mathbf{s}_1, \ldots, \mathbf{s}_k)$ a *global sub-basis* for each $1 \leq k \leq m$. Any global sub-basis $\mathbf{S}_k$ satisfies $\mathbf{S}_k \leq \mathbf{B}$ for all the bases of the solver $\mathbf{B}$ from (5.1). 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 $\mathbf{C} = (\mathbf{c}_1, \ldots, \mathbf{c}_m)$ and performs LLL on the $m + 1$ vectors $(\mathbf{s}_1, \mathbf{c}_1, \ldots, \mathbf{c}_m)$ 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 $\mathbf{S}_k = (\mathbf{s}_1, \ldots, \mathbf{s}_k)$ is sent to the other solvers, which can be merged by LLL on $(\mathbf{s}_1, \ldots, \mathbf{s}_k, \mathbf{c}_1, \ldots, \mathbf{c}_m)$ 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.

5.1.3 Implementation

We introduce new software to realize the strategy of parallel sharing DeepBKZ as described in Subsection 5.1.2. Our software is based on CMAP-LAP [Tat+21], 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 [Ral+18] 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 5.1, we show the overall process of parallel sharing DeepBKZ in CMAP-DeepBKZ. We describe each process in Figure 5.1 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.

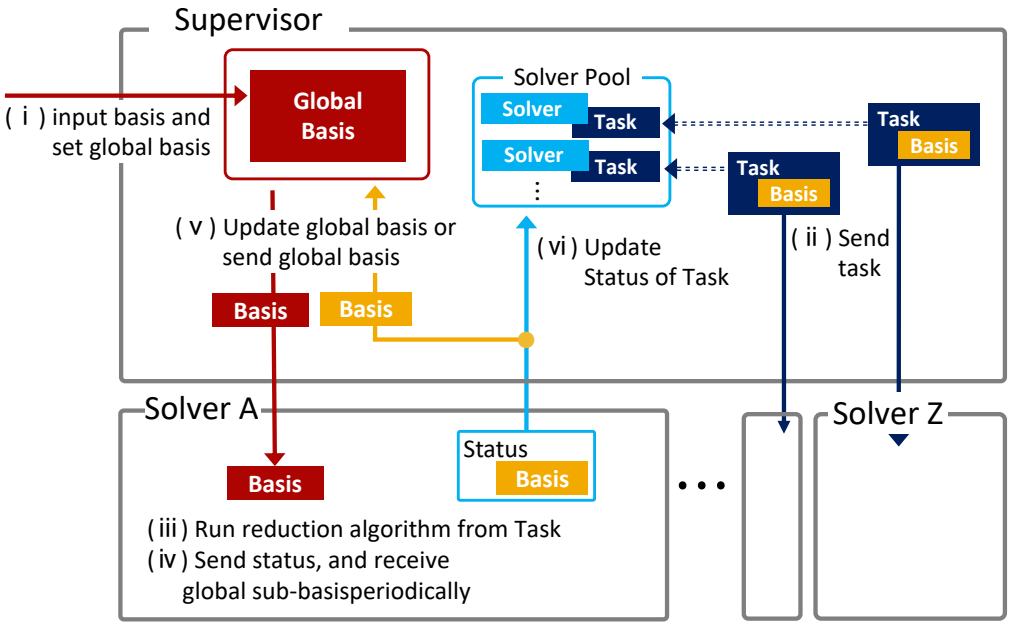

FIGURE 5.1: The overall process of parallel sharing DeepBKZ in CMAP-DeepBKZ

(iv) Every solver sends its status $(\mathbf{B}, \beta)$ to the supervisor periodically, where $\mathbf{B}$ is the current reduced basis and β is the current blocksize of 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.

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 *sampler*s 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

Algorithm 10 Processing flow of the supervisor

```
1: procedure supervisor(B)                                   ▷ B: instance basis
2:     S ← B;                                                ▷ Set initial the global basis S
3:     seed ← 0;
4:     for i = 1 → m do
5:         C ← randomize(B, seed); seed ← seed + 1;
6:         Send task (C, parameters) to i-rank solver;        ▷ Send initial tasks to solvers
7:         SolverPool[i] ← (C, parameters);
8:     end for
9:     while iProbe(source, tag) do
10:        if tag is SolverState then
11:            Receive Status (B, β) from the source-rank solver;
12:                                          ▷ B is basis and β is blocksize of DeepBKZ
13:            Update task of source-rank solver in the solver pool using (B, β);
14:            if B < S_k then
15:                S ← 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 ← randomize(B, seed); seed ← seed + 1;
23:            Send task (C, parameters) to the source-rank solver;
24:            SolverPool[source] ← (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
```

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 [Tat+21], 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.

Processing flow of the supervisor and solver

The pseudo processing flow in the supervisor of CMAP-DeepBKZ is shown in Algorithm 10. 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

Algorithm 11 Reduction algorithm in solver

```
1: procedure Reduction(B, β)
2:
3:     Set t_s to next status sending time;
4:     while Reduction has not finished do
5:         B ← subroutine(B, β);                    ▷ Subroutine of reduction algorithm
6:         if current time > t_s then
7:             Send a status (B, β) to supervisor with SolverState tag;
8:             Wait a notification from supervisor;
9:             if solver receives the global sub-basis S_k = (s_1, ..., s_k) then;
10:                if S_k < B then
11:                    for j = 1 → k do
12:                        l ← minimum index h satisfies ‖π_h(s_j)‖ < ‖b*_h‖;
13:                        B ← LLL((b_1, ..., b_{l-1}, s_j, b_l, ..., 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
```

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 11. 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 11, 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 [Tat+21].

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 CMAP-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 [DG96], 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.

5.2 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.

5.2.1 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 β is an algorithm to find a basis whose i-th basis vector $\mathbf{b}_i$ which is the shortest from the projected lattice $\mathcal{L}\left(\mathbf{B}_{[i,\min(i+\beta-1,d)]}\right)$ for all i. It is natural to compare the projected lattices $\tilde{B}_i := \mathcal{L}\left(\mathbf{B}_{[i,d]}\right)$ and $\tilde{C}_i := \mathcal{L}\left(\mathbf{C}_{[i,d]}\right)$ 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 [BN02]. Denote the singular value decomposition (SVD) of $Y^i(\mathbf{B})Y^i(\mathbf{C})^T$ by

$$Y^i(\mathbf{B})Y^i(\mathbf{C})^T = U\mathrm{diag}\left(\cos\theta_1, \ldots, \cos\theta_m\right)V, \tag{5.2}$$

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 $\theta_1, \ldots, \theta_m \in [0, \pi/2]$ are called the *principal angles* of $Y^i(\mathbf{B})$ and $Y^i(\mathbf{C})$ [BG73; GVL96]. The first principal angle θ_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 $u_1 \in \mathrm{Span}(\tilde{B}_i)$ and $v_1 \in \mathrm{Span}(\tilde{C}_i)$, the second principal angle θ_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 $d\left(Y^i(\mathbf{B}), Y^i(\mathbf{C})\right) = \sqrt{\sum_i \theta_i^2}$. 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:

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

It is shown in [EAS98, 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 θ_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

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

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.

5.2.2 Diversity of bases

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

Definition 5.2.1 (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:*

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

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

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

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

5.2.3 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 (5.1). All solvers share the top-k lattice vectors of the global basis as shown in lines 10–16 of Algorithm 11. The diversity $\mathsf{Div}(\mathcal{B}, d_g)$ 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 5.2 shows the snapshot of $\mathsf{Div}^i(\mathcal{B}, d_g)$ averaged for the five SVP instances after 100 tours of DeepBKZ. We observe that the shapes of $\mathsf{Div}^i(\mathcal{B}, d_g)$ and $\mathsf{Div}^i(\mathcal{B}, d_c)$ 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 $\mathsf{Div}^i(\mathcal{B}, d_g)$ for $i \le k$ 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 $\mathsf{Div}^i(\mathcal{B}, d_g)$ 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 $\mathsf{Div}^i(\mathcal{B}, d_g)$ 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.

For the projection 2-norm, $\mathsf{Div}^i\left(\mathcal{B}, d_{p2}\right)$ 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,

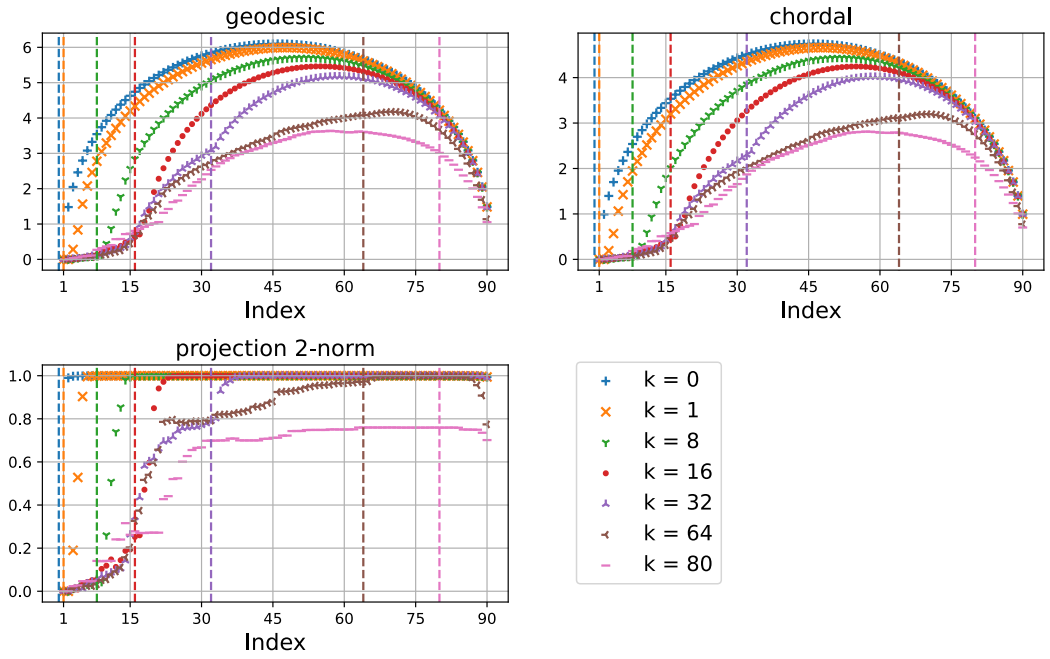

FIGURE 5.2: The average of the i-th projected diversity $\mathrm{Div}^i(\mathcal{B}, d_g)$ computed for 90-dimensional lattice bases with different numbers of shared vectors k right after 100 DeepBKZ tours.)

the lower values of $\mathrm{Div}^i(\mathcal{B}, d_{p2})$ 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 5.3.4.

Transition of the total diversity with tours Figure 5.3 shows the transition of the total diversity $\mathrm{Div}(\mathcal{B}, d_g)$ 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 5.3.4.

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 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.

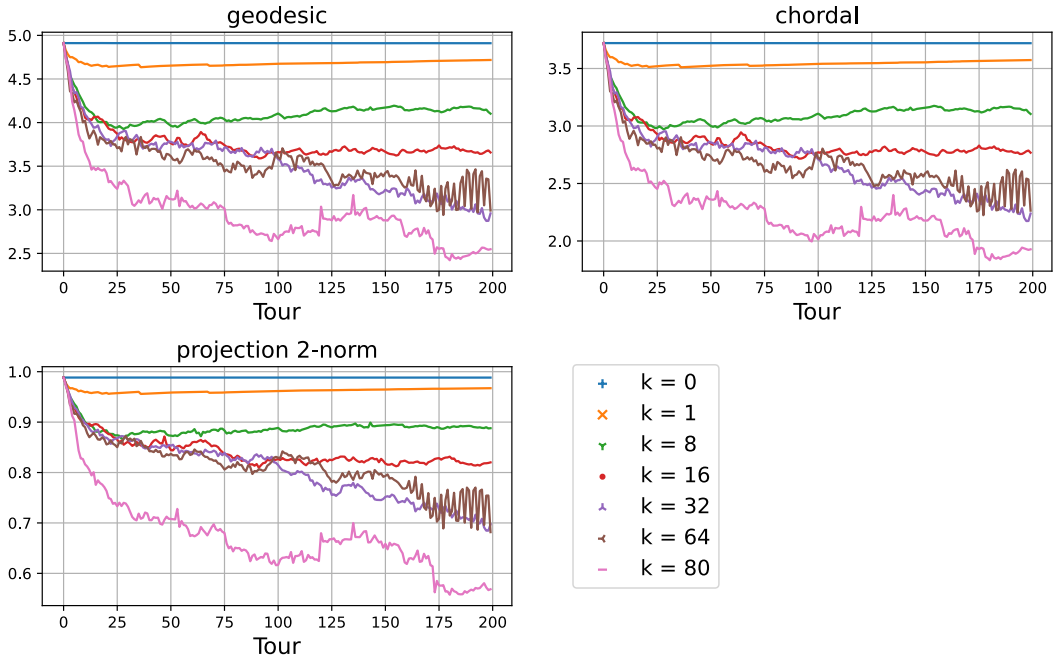

FIGURE 5.3: Transition of the total diversity $\mathrm{Div}(\mathcal{B}, d_g)$ computed for 90-dimensional lattice bases with different numbers of shared vectors k after each tour of DeepBKZ.

First, we generate 100 bases from a single lattice basis by one of the above methods. Then, we calculate $\mathrm{Div}^i(\mathcal{B}, d_g)$ after (i) randomization, (ii) randomization and LLL, and (iii) randomization and a tour of DeepBKZ without sharing. Figure. 5.4 shows the average of $\mathrm{Div}^i(\mathcal{B}, d_g)$ 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 randomization method. It is interesting that the reduction process itself contributes to the diversity of the bases.

Distribution of reduced bases Some lattice algorithms assume the randomness of input bases. For example, *extreme pruning* [GNR10], a pruning technique for enumeration, relies on the heuristic of [GNR10, Heuristic 3] that the normalized Gram-Schmidt vectors $(\mathbf{b}_1^*/\|\mathbf{b}_1^*\|, \ldots, \mathbf{b}_d^*/\|\mathbf{b}_d^*\|)$ 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 [GNR10, Heuristic 3].

Note that we can sample uniformly from $Gr(1, n)$ by sampling from the n-dimensional normal distribution with the zero mean and the identity covariance. By sampling m elements independently from $Gr(1, n)$, we obtain an element of $Gr(m, n)$ almost surely. Let $\mathcal{C}_i$ be a multiset of elements in $Gr(d - i + 1, n)$ sampled in this manner. The value

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

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 5.2.3, we generate a multiset of the lattice bases $\mathcal{B}$ by running

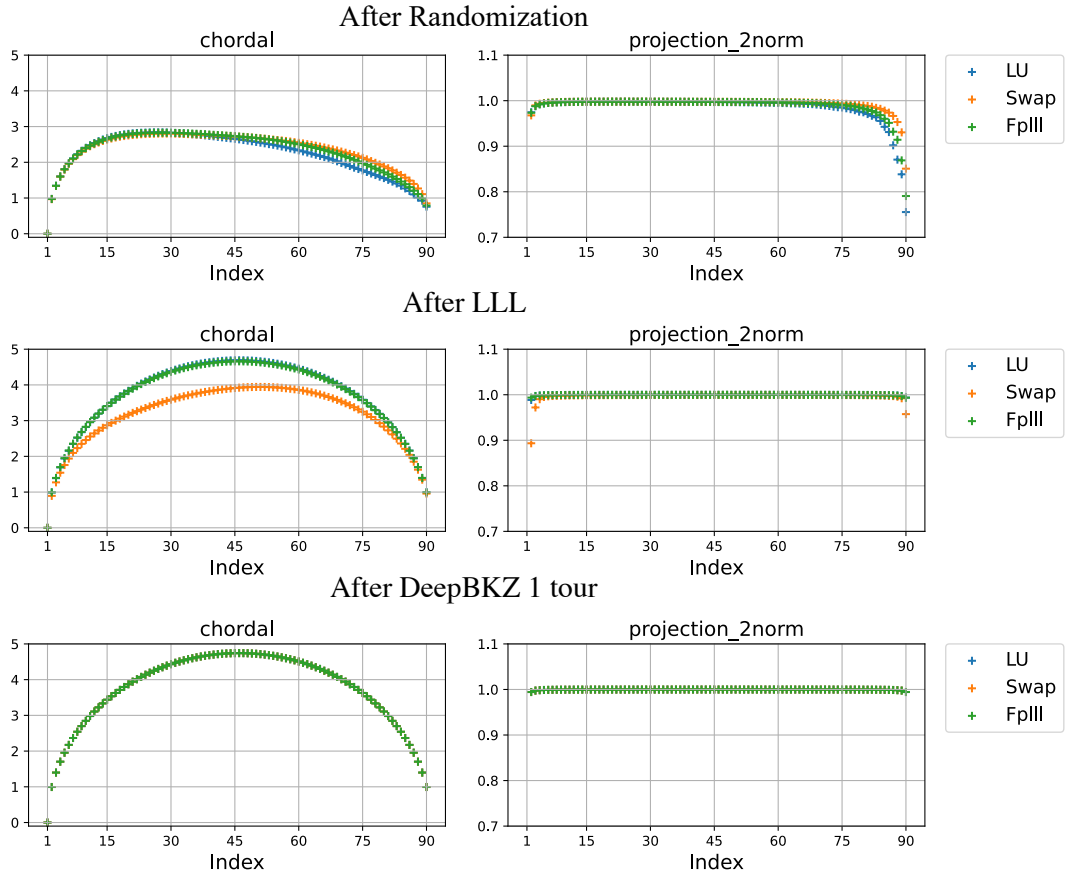

FIGURE 5.4: The i-th projected diversity for the chordal (left) and the projection 2-norm (right) Grassmann metrics computed (top) immediately after randomization, (middle) after LLL, and (bottom) after one tour of DeepBKZ for 90-dimensional lattice bases with different random generation models of unimodular matrices.

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 5.5 shows the difference between $\mathrm{Div}^i(\mathcal{B}, d_g)$ and $\mathrm{Div}^i(\mathcal{C}_i, d_g)$, where $d = 90$ and $|\mathcal{C}_i| = 100$. In addition to the difference of means, the difference between $\mathrm{Div}^i(\mathcal{B}, d_g)$ and the quartiles, denoted by $\mathrm{Div}^i_{25\%}(\mathcal{C}, d_g)$ and $\mathrm{Div}^i_{75\%}(\mathcal{C}, d_g)$, of $\mathcal{C}^i_{d_g} := \{d_g(Y^i(\mathbf{B}), Y^i(\mathbf{C})); (\mathbf{B}, \mathbf{C}) \in P(\mathcal{C})\}$ are shown. We observe that the difference between $\mathrm{Div}^i(\mathcal{B}, d_g)$ and $\mathrm{Div}^i(\mathcal{C}_i, d_g)$ is close to zero. In fact, $\mathrm{Div}^i(\mathcal{B}, d_g)$ fall within the quartiles of the diversity of the random elements $\mathcal{C}^i_{d_g}$ 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 [GNR10, Heuristic 3] holds for bases reduced by DeepBKZ except for the first vector.

5.3 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 5.1 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

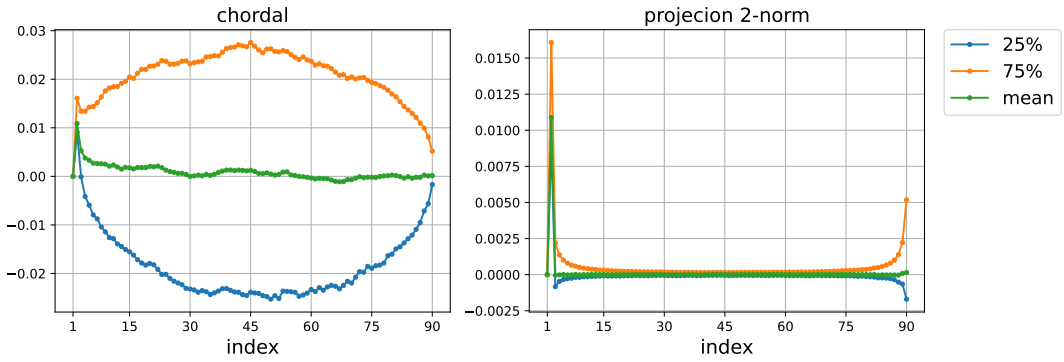

FIGURE 5.5: Comparison between the diversity metrics of $\mathcal{C}$ and that of $\mathcal{B}$. mean: $(i, \mathrm{Div}^i(\mathcal{C}, d_g) - \mathrm{Div}^i(\mathcal{B}, d_g))$, 25%: $(i, \mathrm{Div}^i_{25\%}(\mathcal{C}, d_g) - \mathrm{Div}^i(\mathcal{B}, d_g))$, 75%: $(i, \mathrm{Div}^i_{75\%}(\mathcal{C}, d_g) - \mathrm{Div}^i(\mathcal{B}, d_g))$ for (Left) $d_g = d_c$ the chordal metric, and (Right) $d_g = d_{p2}$ the projection 2-norm.

TABLE 5.1: 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.

experiments, we used instances in the Darmstadt SVP challenge [Sch+10], but we reduced every instance in advance using LLL implemented in the fplll library [The16].

5.3.1 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 $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_d)$ 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 $\gamma = \frac{\|\mathbf{b}_1\|}{\mathrm{vol}(L)^{1/d}}$. As γ is smaller, a reduction algorithm can find a shorter basis vector. Exhaustive experiments in [GN08] 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 $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_d)$ 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 $H_k = \dfrac{R^k \omega_k}{\mathrm{vol}(\pi_{d+1-k}(L))}$ for every $1 \leq k \leq d$ (see [GNR10] for details). As $\mathbf{B}$ is reduced, the total number of nodes N decreases in practice.

- *Geometric Series Assumption (GSA)*: Let $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_d)$ be a reduced basis, and $\mathbf{b}_1^*, \ldots, \mathbf{b}_d^*$ its Gram-Schmidt vectors. The GSA in [Sch03] states that the plots of log-norms $\log \|\mathbf{b}_i^*\|$ of Gram-Schmidt vectors approximate a straight line. (For a β-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 [AD21, Figure 1] for an example of the GSA and its tail-adapted version.) To measure the average quality of $\mathbf{B}$, fpylll [The16] adopts a least squares fit of $\log \|\mathbf{b}_i^*\|^2$ for $1 \leq i \leq d$ is adopted in fpylll [The16] as a slope metric ρ. Under the GSA, the slope relates to the root Hermite factor via $\gamma^{1/d} = \exp\left(-\frac{\rho}{4}\right)$.

5.3.2 Efficacy when sharing short lattice vectors

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

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 5.2.3. 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 5.6, 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 $R = \mathrm{GH}(L)$ as the search radius of enumeration.) Comparing the results for $k = 0$ and $k > 0$, we see that enumeration costs and GSA slopes ρ 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 of basis diversity in a few dimensions.

Analysis of MPI parallelization using CMAP-DeepBKZ

In Figure 5.7 and Table 5.2, 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 5.1 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.)

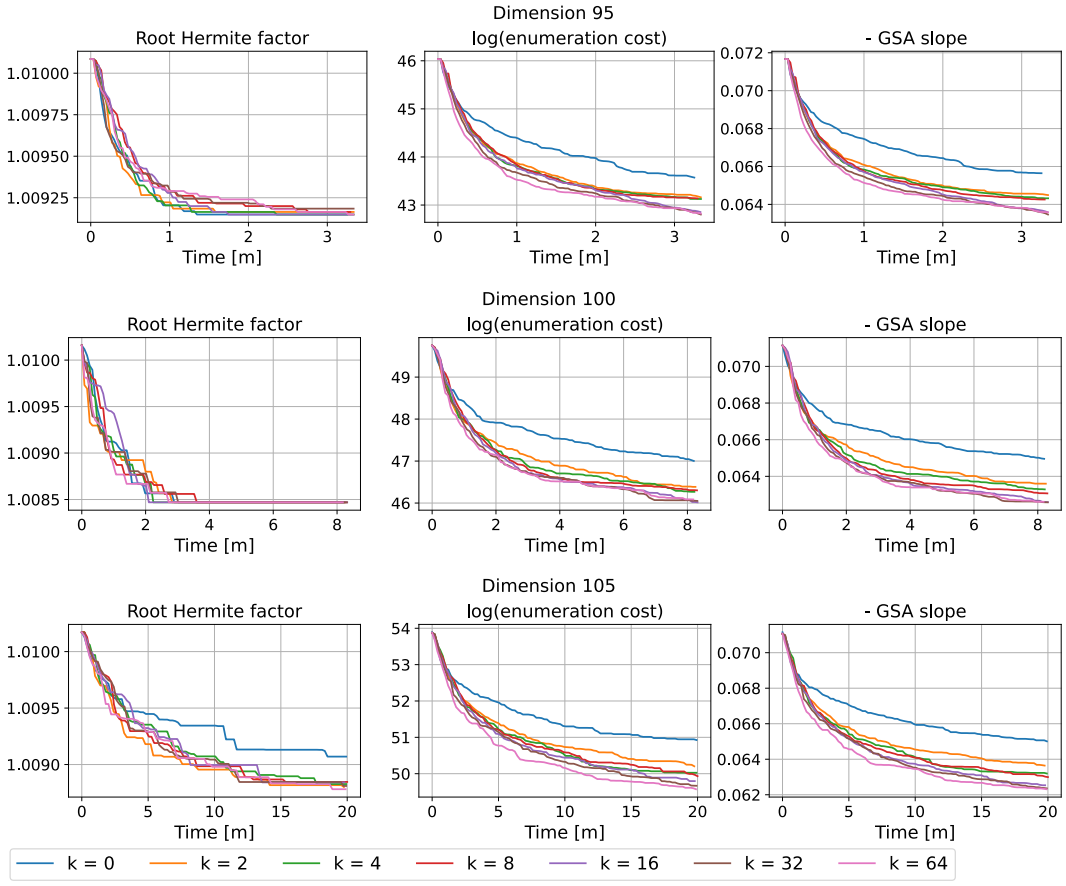

FIGURE 5.6: Transition of metrics on the output quality of parallel sharing DeepBKZ in dimension $d = 95$ (Top), 100 (Middle) and 105 (Bottom), by using $k = 0$, 2, 4, 8, 16, 32 and 64 as the number of short vectors shared among solvers using (Left: the average root Hermite factor $\gamma^{1/d}$, Center: the logarithm of the average enumeration cost $\log(N)$, Right: the minus of the average GSA slope $-\rho > 0$)

In Figure 5.7, 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 5.2, we summarize the experimental results of CMAP-DeepBKZ after six hours of execution. As illustrated in Figure 5.7 and Table 5.2, 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 5.7, 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 5.3.1 (Comparison with BKZ) *In cryptanalysis, BKZ and its variants such as BKZ 2.0 [CN11] are de facto standard reduction algorithms utilized to evaluate the security of lattice-based cryptography (see [Alb+18] for details). Under the GSA and the Gaussian Heuristic, a limiting value of the root Hermite factor of BKZ with blocksize β for a d-dimensional lattice is predicted in [Che13] as*

$$\lim_{d\to\infty} \gamma^{\frac{1}{d}} = \left(\omega_\beta^{-\frac{1}{\beta}}\right)^{\frac{1}{\beta-1}} \sim \left(\frac{\beta}{2\pi e}(\pi\beta)^{\frac{1}{\beta}}\right)^{\frac{1}{2(\beta-1)}} \tag{5.3}$$

for $\beta > 50$ and $\beta \ll d$ (see [Che13; CN11; YD17] for experimental results supporting the prediction). Table 5.2 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 5.3, 5.5 and 5.6 for root Hermite factors of CMAP-DeepBKZ in other dimensions.) In contrast, the prediction (5.3) *implies that BKZ requires around $\beta = 115$ to achieve the same root Hermite factor. Recall that it is the*

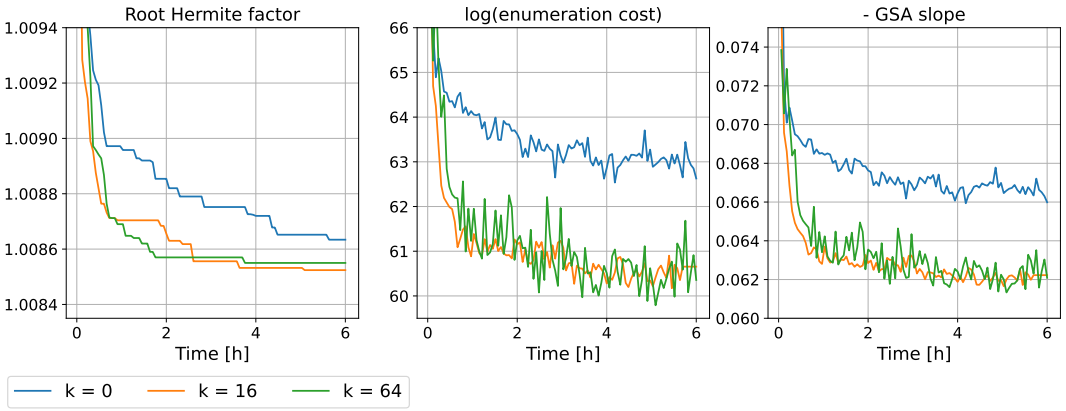

FIGURE 5.7: Same as Figure 5.6, but using CMAP-DeepBKZ and dimension $d = 118$, by using $k = 0$, 16 and 64 as the number of short vectors shared among solvers

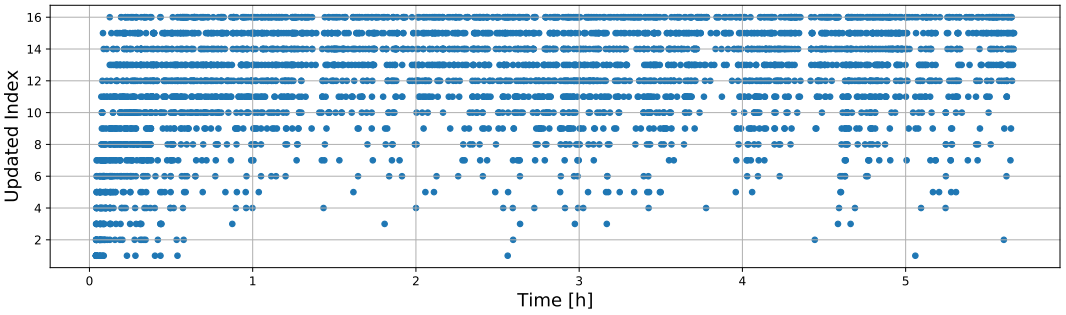

FIGURE 5.8: History of updating a global basis in an execution of CMAP-DeepBKZ with the number of shares $k = 16$ in dimension $d = 118$ (Each plot (x, y) indicates that a global basis at index y was updated at time x)

most dominant factor in both BKZ and DeepBKZ to run an exact-SVP algorithm over projected lattices of dimension β, and the cost is $2^{O(\beta^2)}$ when using an enumeration algorithm for solving exact-SVP in dimension β. Therefore, CMAP-DeepBKZ is significantly more efficient than BKZ without parallelization.

History of updating global bases In Figure 5.8, 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 basis $\mathbf{S} = (\mathbf{s}_1, \ldots, \mathbf{s}_d)$ at index y was updated at time x. We can see from Figure 5.8 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 5.9, we show the approximate factors in projected lattices for a global basis $\mathbf{S} = (\mathbf{s}_1, \ldots, \mathbf{s}_d)$, 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 $y_i = \frac{\|\mathbf{s}_i^*\|}{\mathrm{GH}(\pi_i(L))}$ denotes the approximate factor in the projected lattice $\pi_i(L)$ of dimension $n = d - i + 1$. (Recall that $\mathrm{GH}(\pi_i(L)) \approx \lambda_1(\pi_i(L))$ 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 5.9 that approximate factors at indices $1 \leq i \leq 16$ are extremely close to 1.0 when the numbers of shares $k = 16$ and 64. This implies that the first 16 basis vectors

TABLE 5.2: 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 $\frac{\|\mathbf{b}_1\|}{\mathrm{GH}(L)}$	Root Hermite factor $\gamma^{1/d}$	Machine (Table 5.1)
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	

of $\mathbf{S}$ are almost equal to those of an HKZ-reduced basis. (We also note from Figure 5.9 that $k = 16$ seems sufficient for dimension $d = 118$.)

GSA shapes In Figure 5.10, we show the logarithms of the Gram-Schmidt squared norms $\log \|\mathbf{s}_i^*\|^2$ of a global basis $\mathbf{S} = (\mathbf{s}_1, \ldots, \mathbf{s}_d)$, 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 [Che16] in both cases with and without sharing (cf., see [AD21, 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 5.3.2 (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 5.3.2, the results showed little difference in the root Hermite factor when the number of*

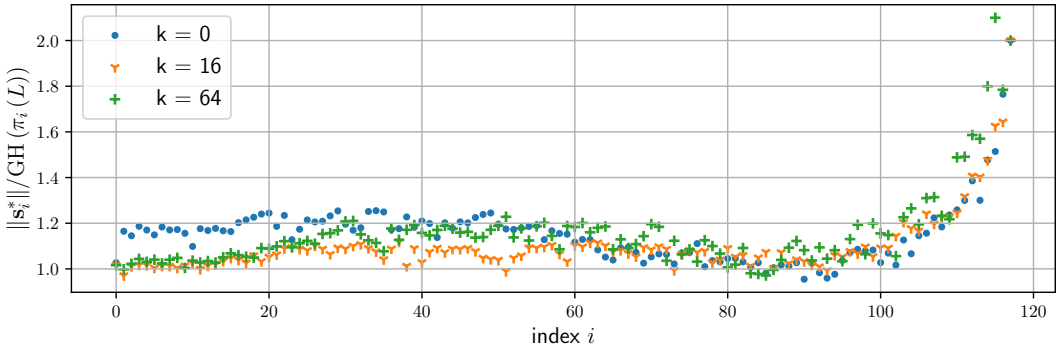

FIGURE 5.9: Plots of approximation factors in projected lattices $\|\mathbf{s}_i^*\|/\mathrm{GH}(\pi_i(L))$ for a global basis $\mathbf{S} = (\mathbf{s}_1, \ldots, \mathbf{s}_d)$ of a lattice L of dimension $d = 118$, output by CMAP-DeepBKZ after 6 hours execution (We used $k = 0$, 16 and 64 as the number of shares in CMAP-DeepBKZ)

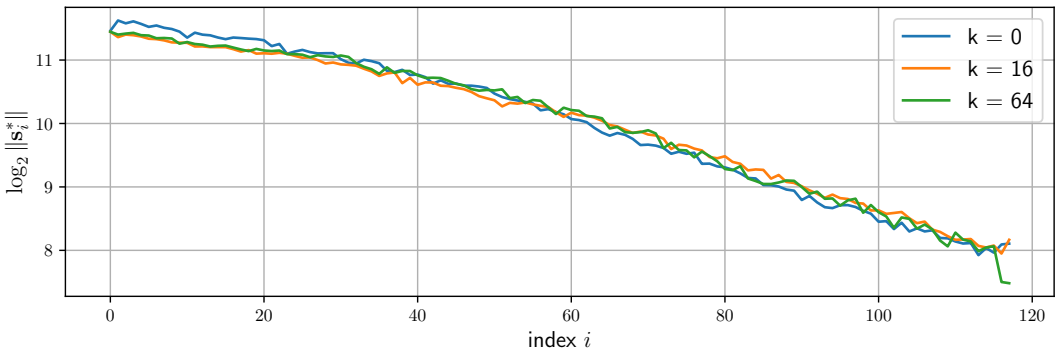

FIGURE 5.10: The logarithms of Gram-Schmidt squared norms $\log_2 \|\mathbf{s}_i^*\|$ of a global basis $\mathbf{S} = (\mathbf{s}_1, \ldots, \mathbf{s}_d)$ output by CMAP-DeepBKZ with the numbers of shares $k = 0$, 16 and 64 after 6 hours execution for an SVP instance in $d = 118$

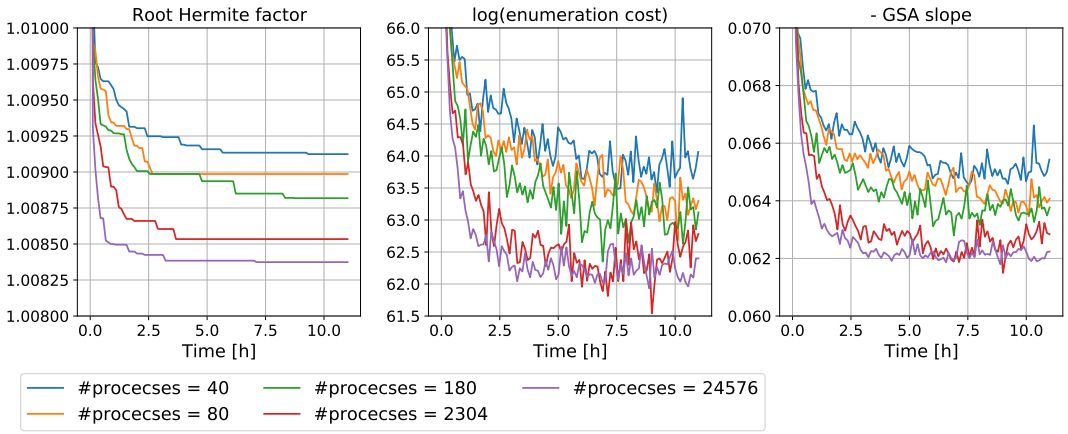

FIGURE 5.11: Same as Figure 5.7, but the dimension is $d = 120$ and lines in each metric represent difference by different numbers of processes (We used $k = 16$ as the number of shares)

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 5.3.2. In addition, as shown in Figure 5.14, 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.

5.3.3 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 $p = 24,576$ cores (see Table 5.3 for details of computing platforms). We ran CMAP-DeepBKZ for 11 hours for every instance of the Darmstadt SVP challenge [Sch+10] 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 β by increments of five with the early termination strategy of [CN11]. (The strategy is also implemented in fplll [The16] 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 5.3 and 5.4, we show experimental results on the scalability of CMAP-DeepBKZ in the dimensions $d = 120$ and $d = 124$, respectively. We assigned one

TABLE 5.3: 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 processes	Updated time [h]	Norm of $\mathbf{b}_1$	Approx. factor $\frac{\|\mathbf{b}_1\|}{\mathrm{GH}(L)}$	Root Hermite factor $\gamma^{1/d}$	Machine (Table 5.1)
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	

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 5.11, we also show the same as Figure 5.7, 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 5.3, 5.4 and Figure 5.11, the quality of a global basis improves in every metric as the number of cores is increased. In particular, Table 5.3 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 α^d lattice vectors of norms less than $\alpha\mathrm{GH}(L)$ 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 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 5.3.

In Figure 5.12 (resp., Figure 5.13), similar to Figure 5.7 (resp., Figure 5.10), 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 5.12 become closer to 1.0 by increasing the number of processes. Similarly, we see from Figure 5.13 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.

5.3.4 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 5.2.2. Figure 5.14 is created from five results of 118-dimensional instances in Section 5.3.2, with six hours executions using 2,304 cores and 16 shared short vectors. Figure 5.14 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 relatively large intervals of 120 seconds, the situation where the top-16 vectors are aligned

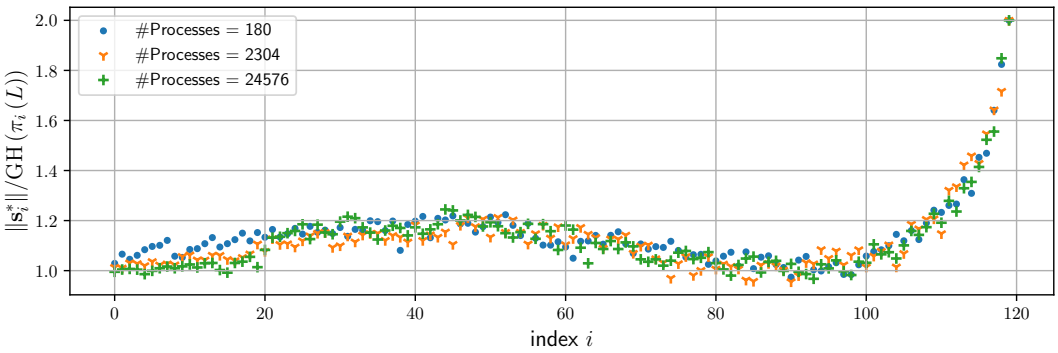

FIGURE 5.12: Same as Figure 5.7, but the dimension is $d = 120$ and plots represent difference by different numbers of cores (We used $k = 16$ as the number of shares)

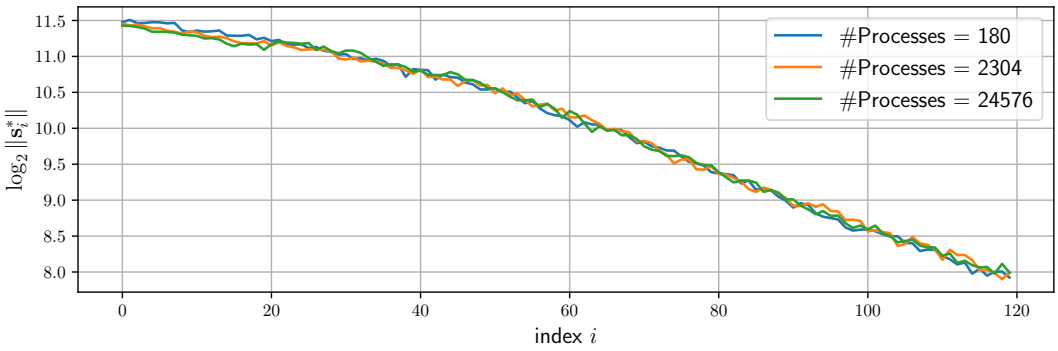

FIGURE 5.13: Same as Figure 5.10, but the dimension is $d = 120$ and three lines represent different GSA shapes by different numbers of processes (We used $k = 16$ as the number of shares)

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 24,576 cores. Figure 5.15 was created from the results of experiments utilizing up to 24,576 cores in 11 hours executions on a 120-dimensional SVP in Section 5.3.3. The figures are the same as Figure 5.14 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.

TABLE 5.4: Same as Figure 5.3, but the dimension is $d = 124$

SVP instance	Number of processes	Updated time [h]	Norm of $\mathbf{b}_1$	Approx. factor $\frac{\|\mathbf{b}_1\|}{\mathrm{GH}(L)}$	Root Hermite factor $\gamma^{1/d}$	Machine (Table 5.1)
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	

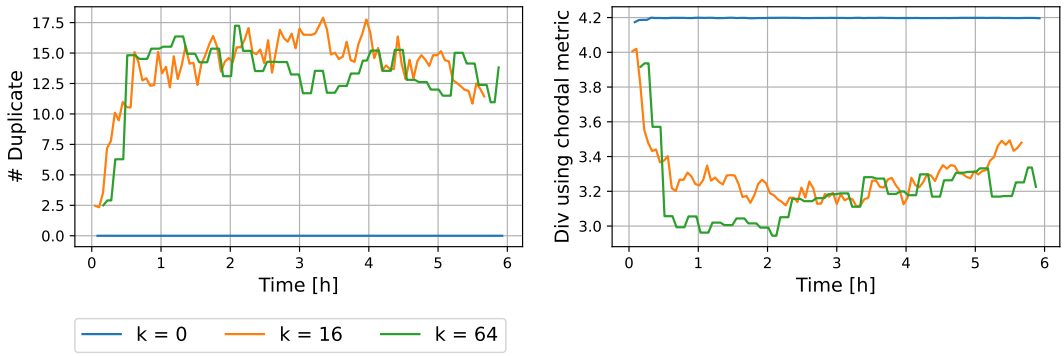

FIGURE 5.14: Transition of the diversity of 118-dimensional lattice basis with different the number of shared vectors; left figure is the transition of the number of overlap of basis vectors, right figure is the transition of the Div with Projection metric

5.3.5 Massive parallelization experiments with checkpoints and restarts

For CMAP-DeepBKZ, we conducted large-scale experiments on the supercomputer systems Emmy and Lisa (Table 5.1) with multiple checkpoints and restarts for instances of the Darmstadt SVP challenges [Sch+10] in dimensions $d = 128$, 130 and 132. In Figure 5.16, 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 5.5, we summarize the final output results of Figure 5.16. 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 5.5. In contrast, it was reported on the webpage of [Sch+10] 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., approximation factor) of which was about 2882 (resp., 1.00477). However, the norms of Table 5.5 in the other dimensions $d = 130$ and 132 do not surpass the current records yet.

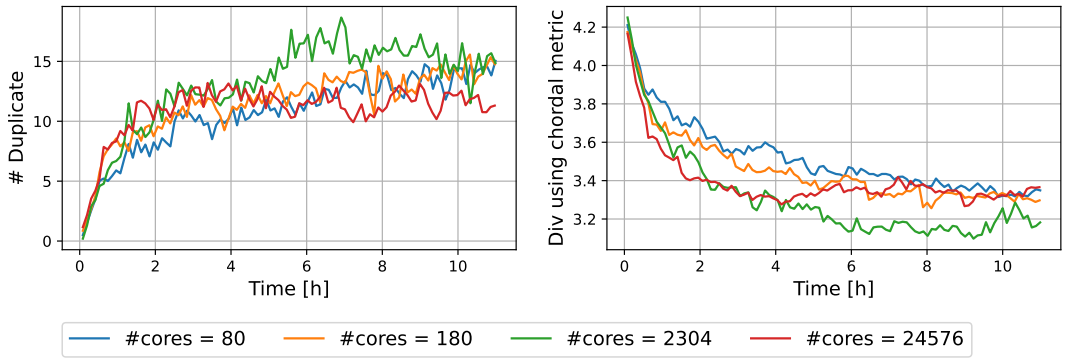

FIGURE 5.15: Same as Figure 5.14, but dimension is 120 and with different the number of cores.

TABLE 5.5: Large-scale experimental results of CMAP-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	Updated	Norm	Approx.	Root Hermite	Machine*
Dim.	Seed	cores*	time [h]	of $\mathbf{b}_1$	factor $\frac{\|\mathbf{b}_1\|}{\mathrm{GH}(L)}$	factor $\gamma^{1/d}$	(Table 5.1)
128	1†	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

† a new solution for the Darmstadt SVP challenge [Sch+10] in dimension 128 (see also Table 5.6 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.

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 5.1 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 5.16). In Figure 5.17 and 5.18, we show snapshots of a global basis $\mathbf{S} = (\mathbf{s}_1, \ldots, \mathbf{s}_d)$ 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 $\|\mathbf{s}_i^*\|/\mathrm{GH}(\pi_i(L))$ 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 5.18. 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 ρ of the entire basis is -0.05867, but the ρ of the sub-basis consisting of $(\mathbf{s}_1, \ldots, \mathbf{s}_{32})$ 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 $p - 96 = 103,524$ 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 5 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.

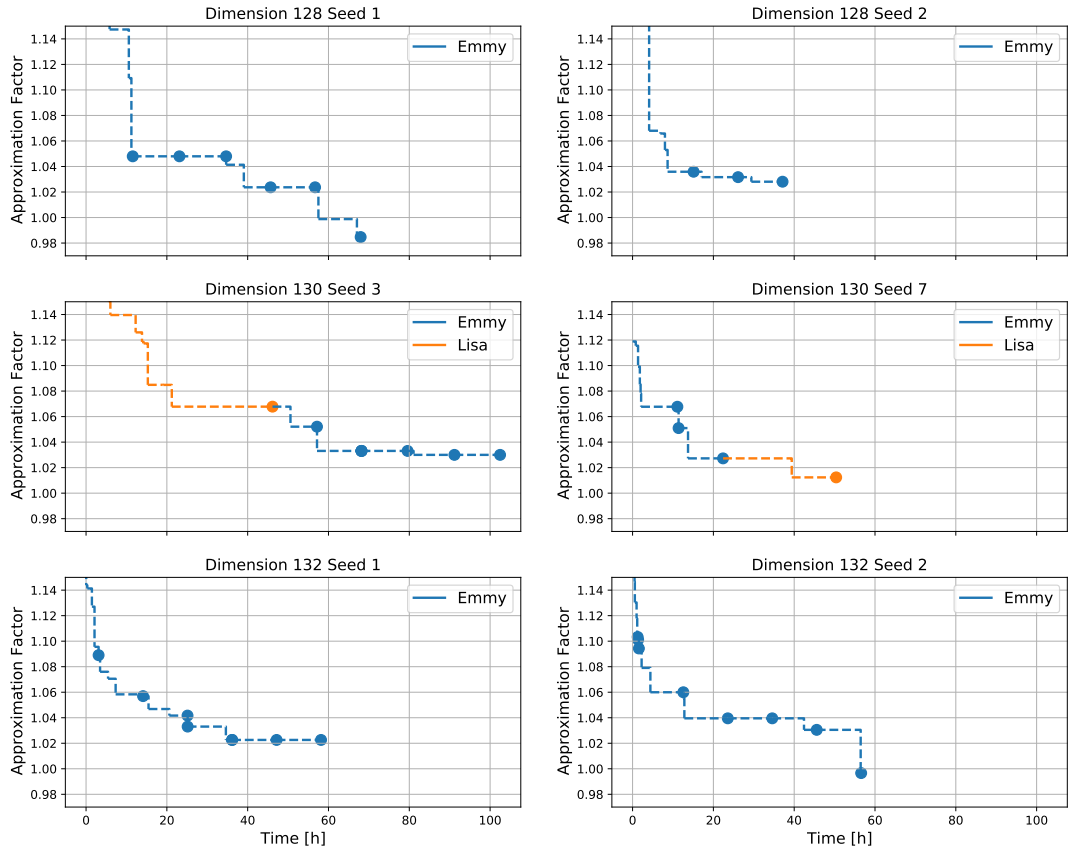

FIGURE 5.16: Transition of the approximation factor $\frac{\|\mathbf{b}_1\|}{\mathrm{GH}(L)}$ of a shortest basis vector $\mathbf{b}_1$ for SVP instances in dimensions $d = 128$, 130 and 132 (Each dot show the timing of checkpoint-and-restart, and see also Table 5.5 for a summary)

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 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.

New solutions for the Darmstadt SVP challenge In Table 5.6, we list new solutions proposed by CMAP-DeepBKZ for the Darmstadt SVP challenge [Sch+10]. For each dimension $d = 103$, 105, 107, 109, 113 and 114, we performed CMAP-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

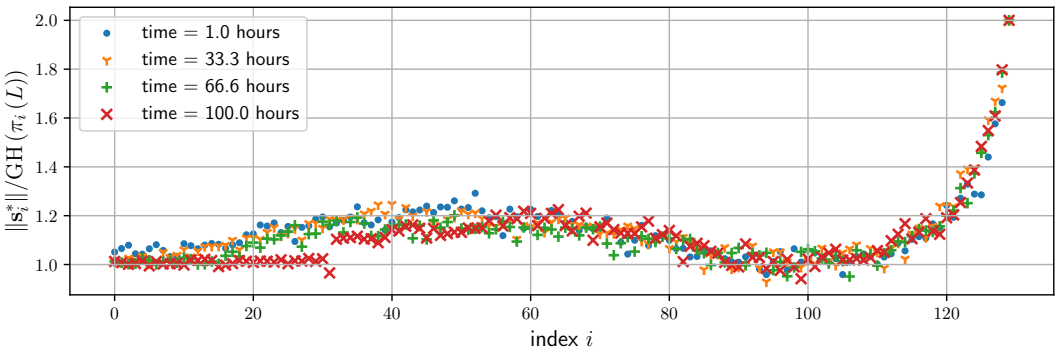

FIGURE 5.17: Plots of approximation factors in projected lattices $\|\mathbf{s}_i^*\|/\mathrm{GH}(\pi_i(L))$ for a global basis $\mathbf{S} = (\mathbf{s}_1, \ldots, \mathbf{s}_d)$ output by CMAP-DeepBKZ of a lattice L of dimension $d = 130$ with seed $= 7$ of SVP challenge instance after $1.0, 33.3, 66.6, 100$ hours executions, and the final numbers of shares $k = 32$

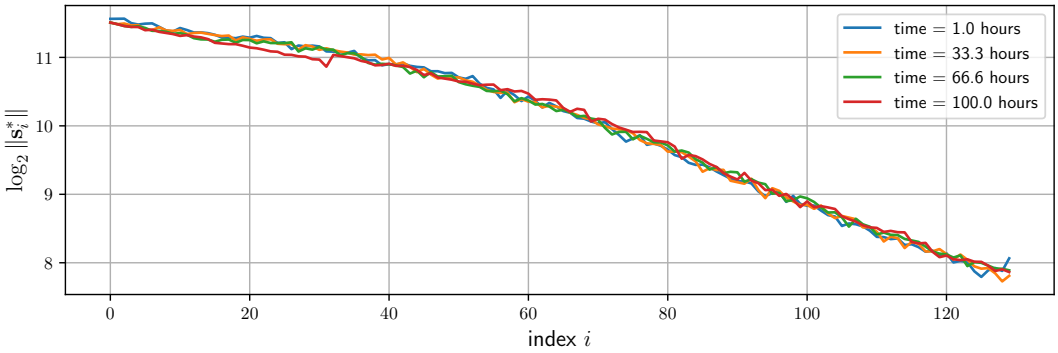

FIGURE 5.18: The logarithms of Gram-Schmidt squared norms $\log_2 \|\mathbf{s}_i^*\|$ of a global basis $\mathbf{S} = (\mathbf{s}_1, \ldots, \mathbf{s}_d)$ of a lattice L same as Figure 5.17.

executing five instances with seeds ranging from zero to four (resp., two instances of seeds 1 and 2) on Emmy. In [Tat+20], 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 [Tat+20, Tables II and III]). In contrast, Table 5.6 shows that it took about 57.5 hours for $d = 128$ by CMAP-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 [Alb+19], the state-of-the-art SVP solver using advanced sieve algorithms as described in Subsection 1.3. G6K adopts the sub-sieve strategy of [Duc18]. 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 [Alb+19, 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 [DSW21, 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

TABLE 5.6: New solutions for the Darmstadt SVP challenge [Sch+10], found by parallel sharing DeepBKZ with the number of shares $k = 16$

SVP Instance		# of	Updated	Norm	Approx.	Root Hermite	Machine
Dim.	Seed	cores	time	of $\mathbf{b}_1$	factor $\frac{\|\mathbf{b}_1\|}{\mathrm{GH}(L)}$	factor $\gamma^{1/d}$	(Table 5.1)
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

TABLE 5.7: Same as Table 5.6, but $k = 1$

SVP Instance		# of	Updated	Norm	Approx.	Root Hermite	Machine
Dim.	Seed	cores	time	of $\mathbf{b}_1$	factor $\frac{\|\mathbf{b}_1\|}{\mathrm{GH}(L)}$	factor $\gamma^{1/d}$	(Table 5.1)
104	35	120	551 s	2516	0.97173	1.00872	CAL A
	85	120	214 s	2520	0.97010	1.00870	CAL A
	82	120	432 s	2529	0.97719	1.00877	CAL A
111	8	2,000	792 s	2597	0.96979	1.00866	ITO
	30	2,000	541 s	2635	0.98382	1.00856	ITO
	29	2,000	611 s	2660	0.99467	1.00843	ITO
121	2	2,304	682 m	2780	0.99706	1.00840	ITO
	4	2,304	481 m	2809	1.00820	1.00830	ITO

d with G6K in the maximum sieving dimension m. As shown in Tables 5.5 and 5.6, the performance of CMAP-DeepBKZ in dimensions around $d = 130$ is faster than that of G6K around $m = 130$ in [Alb+19, Table 2] if we ignore the difference of computing resources. In contrast, the performance in [DSW21, 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 [DSW21] 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 β, and its space-complexity is polynomial with respect to β. In particular, CMAP-DeepBKZ has sufficient performance even with small blocksizes 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.

Chapter 6

Conclusion

We proposed a framework for lattice problems, and a solver for SVP. Lattice problems are a type of discrete optimization problem that is difficult to solve, even for a quantum computer. There is little research on solving this problem in large-scale distributed systems. In addition, the difficulty of solving the lattice problems supports the security of major cryptographic systems in post-quantum cryptography. Therefore, investigating the potential of large-scale parallel computation of the lattice problems is important in the field of optimization and cryptanalysis.

In Chapter 4, we propose a novel large-scale framework, CMAP-LAP, for lattice problems. CMAP-LAP offers a multi-algorithm paradigm in which multiple types of lattice algorithms run in parallel while sharing information to improve the performance of the entire system. To realize this paradigm, we have developed four key components. Our communication interface class enables hybrid parallel processing, independent of the solver's internal algorithms. This makes it easy to incorporate existing solvers, those run not only on shared-memory systems but also on distributed-memory systems [Mun+19]. The efficient collection and distribution of short lattice vectors by the management process facilitate information exchange among heterogeneous solvers. This is based on the fact that each lattice algorithm generates short lattice vectors as by-products, which can be utilized by other algorithms if shared. Furthermore, the management process generates new tasks from the collected information and assigns them to the solvers in order of the estimated likelihood of finding a solution. The periodic collection of all solvers's progress by the management process allows the grasp of the overall system status. This is used to adjust the assignment of tasks to solvers. In addition, a powerful checkpoint functionality is implemented, which is essential for long execution times. The management of memory and communication delays is carefully realized, which are essential for the stability of large-scale parallel execution. Several numerical experiments demonstrated the stability, scalability, and checkpointing of CMAP-LAP and showed performance improvement through information sharing and heterogeneous execution of multiple algorithms.

In Chapter 5, we propose software CMAP-DeepBKZ using the framework of CMAP-LAP for massively parallel execution of a reduction algorithm of BKZ-type. Our software enables us to simultaneously execute a reduction algorithm on randomized bases by sharing short basis vectors among solvers in order 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 cannot be almost lost during the execution of parallel reduction with sharing $k \leq 64$ short basis vectors for high-dimensional lattices (Figures 5.2, 5.3 and 5.14). Furthermore, we demonstrated by experiments that sharing $k = 16$ short basis vectors is effective in both the output quality and the performance of CMAP-DeepBKZ that is our software in using DeepBKZ [YY17] as a reduction algorithm. Our experiments (Table 5.5) showed that CMAP-DeepBKZ with small blocksizes 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 with up to $103,680$ cores, without using any strategy like the sub-sieve of [Duc18] adopted in G6K [Alb+19]. In particular, it took about 57.5 hours using 24,576 cores to find a new solution of the Darmstadt SVP challenge in dimension $d = 128$ (Table 5.6).

The framework and software this thesis have proposed and implemented are based on UG and are part of its derived applications. Figure 6.1 shows the list of frameworks and software based on UG is shown in the figure. It is expected that further high-performance SVP solvers will be developed based on the CMAP-LAP and CMAP-DeepBKZ.

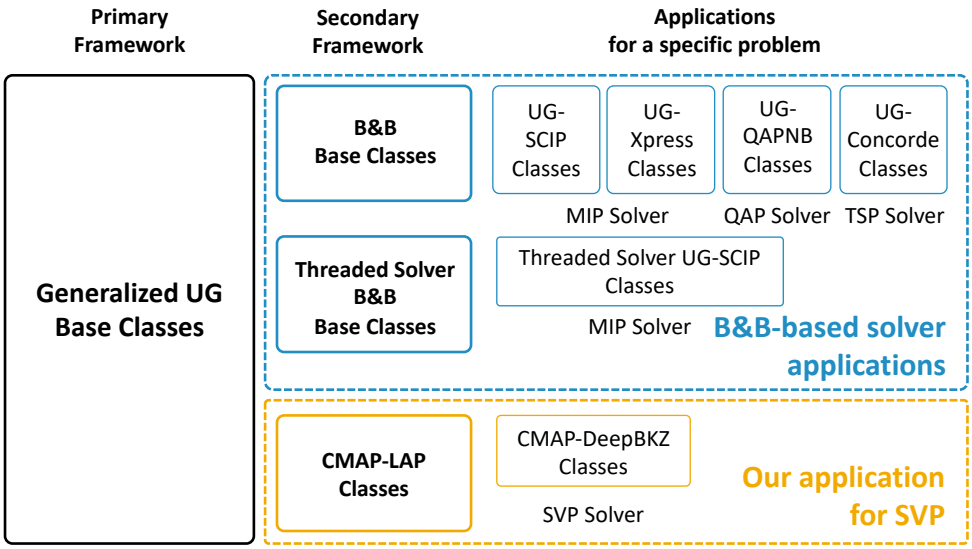

FIGURE 6.1: Frameworks and applications based on Generalized UG

Acknowledgements

First of all, I would like to express my greatest appreciation to Professor Katsuki Fujisawa at Kyushu University for his continuous useful comments and encouragement. In addition to his lectures on optimization, HPC, and so on, he has given me opportunities for joint research with companies and presentations, and his high-spec and large-scale computing environment has provided me with many skills and knowledge necessary for my research activities. I would like to express my gratitude to Dr. Yuji Shinano at Zuse Institute Berlin (ZIB) for his significant contribution to the framework and software development, including implementation techniques for parallel processing and executions on supercomputers day and night. I would also like to thank Associate Professor Masaya Yasuda at Rikkyo University for taking care of my research, valuable discussions, and various insights. The basic knowledge of lattices and helpful pseudo-codes greatly supported my research. I am grateful to Professor Shizuo Kaji at Kyushu University for his knowledge of mathematics and implementation from various viewpoints. I would also like to thank Fujisawa ' s laboratory staff, Mrs. Tomoko Sakai and Mrs. Kyoko Ikebe, for their support in all other aspects and past and present members of Fujisawa ' s laboratory for spending a pleasant time with me. Finally, I would like to express my gratitude to my family and friends for their support and warm encouragement.

Appendix A

Solutions of SVP Challenge

A.1 New records in the hall of frame of SVP challenge

Through the development of CMAP-LAP framework and CMAP-DeepBKZ software, we have updated the hall of frame of SVP challenge in dimensions 128, 127, 124, 121, 113, 111, 109, 104, 103 and 96. The records are shown in Table A.1.

A.2 Solutions closed to record in the hall of frame of SVP challenge

We have also found vectors in Table A.2 that are close to the hall of frame of the SVP challenge, although the CMAP-LAP framework and CMAP-DeepBKZ software have not updated the record. These are also used to evaluate the performance of these framework and software as well.

TABLE A.1: New records in the hall of frame of SVP challenge

SVP Instance		norm	Found vector
Dim.	Seed		
128	1	2812.00	[-43 -272 -328 267 -121 123 -308 301 116 99 -96 22 -13 -185 317 286 227 -155 -58 75 -176 -283 -184 524 -271 -259 696 -288 -422 193 -91 -146 -568 -174 305 -11 282 -220 17 -82 64 427 8 -144 -105 8 123 169 -274 128 -395 -30 264 31 230 328 -56 51 -287 397 -368 -108 -35 30 90 90 -171 -123 56 239 123 556 -628 140 -255 -28 -205 20 -180 -274 -115 -89 -19 -164 35 53 -401 428 -25 -90 -240 59 -275 -112 -71 -259 -55 -175 -461 36 -271 -382 -313 12 718 -127 123 -145 -253 440 146 256 155 151 374 6 34 -201 151 74 -227 82 289 -508 -29 365 -184 103]
124	2	2826.79	[59 99 -145 383 38 -140 188 33 -493 -58 9 82 -92 -594 -249 12 23 34 260 -356 577 3 292 197 -207 144 -585 382 267 158 -188 -158 -579 -115 -40 -361 51 93 155 -174 -49 -308 170 241 253 183 1 -224 231 286 169 295 -332 -287 495 15 -37 -230 116 -214 -90 96 375 -331 76 -401 71 42 -436 431 -111 -21 121 -57 -38 -712 190 104 235 -102 234 -70 -2 432 -93 -242 42 -191 -238 474 -113 -234 -29 315 -539 289 122 -139 -64 249 -169 -294 63 286 161 -82 -198 17 86 -15 51 550 112 26 -460 155 30 -250 341 -203 -164 -320 -295 139]
121	4	2780.01	[50 -79 -316 67 138 398 -259 -258 12 -401 -164 -172 141 13 -26 -222 349 427 105 -35 75 611 61 466 25 -112 157 16 340 252 93 14 332 -225 165 -411 -202 -29 -52 -90 -255 9 -290 224 97 -17 -167 114 419 -20 169 380 134 -367 -63 -185 208 -94 -167 -434 429 110 273 -421 -17 -133 -74 4 -191 175 -91 -80 105 78 481 -147 816 -254 -178 -15 -162 -132 -93 -130 -164 462 -154 336 386 -229 9 159 -2 92 -516 97 -200 67 465 46 293 115 11 340 -364 189 177 422 113 441 -223 -302 -223 291 26 -182 105 205 556 -358 50]
121	2	2809.23	[94 435 -130 -168 -158 -86 -334 165 -20 166 245 422 370 317 185 125 -431 -175 -31 -64 57 -126 -173 348 282 -81 158 -480 -468 110 -238 -160 -216 -337 235 450 555 -279 358 355 294 563 110 81 24 -105 -154 -238 254 228 -496 36 181 215 187 -70 -132 -156 189 37 -34 124 112 281 -427 -283 -80 -245 222 73 -1 -64 502 -261 177 -366 409 25 88 164 330 -88 -402 12 120 -188 128 322 -206 -186 -250 -55 -436 -123 -112 -215 231 -359 -186 75 54 118 220 486 -350 41 -192 258 -45 -38 253 -76 -416 107 393 488 301 -138 418 140 -8]
113	5	2621.54	[436 -217 542 327 7 25 162 352 15 -38 -311 -166 -126 -33 223 -154 -5 -19 -242 216 -63 401 -112 115 230 -191 146 -43 67 258 -110 -524 -74 87 -249 89 96 -133 53 -147 56 -157 58 168 -129 82 -2 177 -273 -22 429 -489 137 232 -93 -246 198 365 -303 -284 -214 113 -556 -23 47 269 -407 -53 279 862 -356 232 22 251 44 -420 37 -229 493 -282 -199 -336 72 404 -39 -161 180 194 644 -114 436 29 178 -157 -46 -22 41 130 -80 -223 -311 -3 49 52 480 55 -93 -8 -74 246 88 -211 63]
111	29	2597.33	[-235 396 -2 69 9 547 418 106 -124 72 -81 113 103 -107 -240 -532 387 -178 104 157 504 -38 -88 63 -198 -261 -48 72 265 126 39 -102 212 -99 249 376 -7 19 125 -44 -59 173 -56 139 -385 -200 59 136 3 178 109 431 -538 -290 38 -308 71 196 17 176 35 -313 -25 550 1 -9 261 -137 487 262 65 80 -459 -30 733 -361 91 -459 511 -64 284 -40 103 -34 520 -480 -22 317 -153 167 -125 306 -90 31 -154 -257 -62 -88 118 -177 257 398 -35 332 -143 91 -110 -122 94 -16 -216]

SVP Instance		norm	Found vector
Dim.	Seed		
111	30	2635.23	[-650 -123 146 71 61 93 12 -564 251 -316 143 299 -273 -347 -144 120 -212 -130 166 94 142 21 -183 411 -286 -146 112 -139 57 -175 288 -50 -8 76 254 -28 -365 -352 352 -205 -709 -240 -82 81 147 -173 -8 66 -81 -161 -266 179 338 -135 90 -197 91 456 7 -293 -59 -194 338 -191 245 84 -179 -156 -221 208 -145 -41 -92 -68 353 302 -34 -25 32 367 -172 187 -266 473 -228 156 -280 -491 131 -188 161 350 -333 -291 -463 180 -126 -417 458 -565 -27 227 96 -258 -159 -201 -62 68 -131 183 -406]
111	8	2660.27	[124 362 95 -172 -335 375 -127 -153 -324 141 -221 -147 416 -4 -275 262 403 -544 494 272 -63 -182 -267 161 -120 60 93 327 -189 -253 79 -223 -60 24 -130 132 -245 349 668 -124 -226 432 -83 -364 150 -14 -292 182 -70 -342 148 -21 78 -241 363 437 242 -98 142 -14 -1 -302 166 70 -217 -195 26 -24 196 -36 -157 -367 -472 205 116 353 553 -37 -166 233 -81 -279 -118 133 -22 24 -115 287 -437 -144 -129 -402 -18 126 134 207 304 21 -67 -218 106 -254 -713 -283 -236 -112 -259 -28 -452 -190 -33]
109	2	2559.17	[73 -110 -44 -74 -19 451 183 -138 -68 61 230 -113 -450 178 -31 -253 5 -110 14 -385 -169 -205 440 17 -432 318 -197 -138 -344 46 428 483 142 -86 377 -223 -179 -25 28 -321 245 -165 12 -152 -53 -249 -90 -211 139 249 -323 -266 -84 -862 -440 201 250 96 -124 -485 -346 -25 61 -158 -382 492 100 -180 -132 -193 31 -19 105 134 -271 -18 -542 59 -7 -232 118 -127 -160 19 132 22 131 149 183 159 77 96 675 37 152 -457 -137 121 -256 239 -10 85 50 419 43 -299 -192 -52 56]
104	35	2516.02	[41 419 -285 -313 75 -397 -106 202 413 281 81 23 -57 -262 168 -103 199 168 -192 -265 113 94 157 116 -427 611 -132 140 297 -338 -206 -500 -59 -74 -367 433 175 -349 105 -263 18 -32 355 399 320 115 -60 -482 -175 -34 -266 -161 407 360 -236 0 -414 86 -162 5 -89 -180 556 139 -216 -113 -178 165 88 7 245 -71 333 -83 -115 102 -263 -65 477 61 20 -97 276 52 -8 108 189 -253 -260 -669 406 -114 83 124 17 19 -88 243 209 -117 184 68 299 93]
104	85	2520.11	[-334 -448 -108 -285 -251 39 -104 138 84 -46 260 -62 -136 -142 36 -237 222 362 724 -55 227 -165 163 -41 -173 -182 -201 206 -338 182 202 -249 271 130 -113 -136 193 -271 -256 -168 -169 -227 426 261 -416 73 114 -10 -203 -107 70 -392 -59 -70 -558 -383 448 69 -28 -220 230 -197 -54 -180 -30 -322 -231 1 -433 159 259 19 291 237 616 -277 -291 -88 -179 -261 28 157 237 -73 -26 -325 -140 -29 -109 161 -117 314 -257 195 -220 -144 793 -268 124 241 113 -27 -62 -36]
104	82	2529.01	[20 -79 -397 -436 213 61 -57 307 -179 376 13 74 68 26 260 226 146 56 267 387 -789 244 -129 -130 402 -11 -70 278 54 188 90 69 19 317 -37 133 -17 -9 200 160 143 -302 133 -90 -460 -374 -7 271 -88 75 -373 -246 -142 -259 401 -57 111 -126 189 353 258 -55 -81 -416 133 -498 -44 -150 -10 146 271 -375 1 -324 -147 -85 93 -37 62 221 463 -342 69 -279 54 -114 477 -434 -156 391 56 756 -97 127 80 -199 2 43 332 -66 -364 30 -199 -69]
103	3	2507.58	[170 -185 -274 393 -201 97 122 -220 -86 233 490 -128 160 -348 255 -89 -254 449 97 -1 197 -521 -88 -34 -87 -29 -70 -185 -125 -110 -91 -378 -100 -95 72 55 -59 216 -190 -15 264 41 -126 -213 697 82 229 232 -240 376 160 14 220 -353 555 -156 422 -193 -320 293 166 -180 -81 392 -182 -290 -167 -15 -222 200 -458 60 -105 217 -29 81 -241 -169 790 57 271 121 125 -104 -23 19 -98 -77 360 69 -489 -327 81 61 113 -219 -239 -24 18 182 -418 187 337]

SVP Instance		norm	Found vector
Dim.	Seed		
96	18	2440.53	[248 -276 29 -351 316 362 226 -255 -372 236 -443 76 -465 -36 230 -274 -447 -137 -68 -52 166 614 36 -589 -54 70 -165 -89 276 -279 53 -68 97 -138 116 249 -249 -256 6 -305 304 -122 -549 162 -69 67 549 -5 -308 527 178 172 204 171 -171 65 -118 -35 -208 154 -245 -22 170 71 -16 -402 361 -122 -232 487 -268 349 -9 100 -199 78 -329 -139 290 288 -130 -247 -218 26 223 -238 -341 -113 -92 -219 -158 147 -98 97 -70 86]
96	14	2484.15	[195 -397 51 -195 171 -37 73 209 101 -172 243 196 -70 220 -65 305 -259 24 497 -51 -110 -160 -164 67 -446 363 28 -503 229 -125 -117 43 -60 10 -78 -58 -35 -93 -35 -506 -147 -126 265 97 -498 139 227 78 -269 -213 -131 -301 -346 24 379 -319 -53 95 -438 -387 -135 50 160 313 -403 -410 183 -76 -32 509 96 -174 -467 138 -44 131 540 -64 -478 381 -203 348 498 340 136 -353 -215 238 -326 -111 -150 -70 -152 5 -322 249]
96	19	2489.60	[55 -93 -50 -50 -426 445 -198 -362 -167 390 114 411 10 -12 18 87 -118 -354 210 -1 -10 -112 -252 225 -286 250 5 276 -385 543 -64 -80 -120 -511 -386 380 -314 -292 237 248 56 141 233 16 -98 -312 -106 -238 288 594 151 -321 -154 -154 -88 33 -11 -184 -496 432 344 10 253 -156 563 135 -79 384 154 31 -65 -239 -631 16 -33 -70 -8 -49 217 87 303 -134 -103 43 204 -396 175 265 -556 173 -75 -34 57 167 -155 -66]
96	7	2495.95	[-251 -150 -79 -74 -347 -25 -370 388 -30 295 170 -167 216 404 591 -117 -332 -431 85 -356 -512 -78 -80 -72 -45 448 224 -9 -282 -81 64 345 356 -312 294 -156 81 191 160 3 334 323 128 224 242 -174 -92 82 436 -30 -53 209 365 -194 -400 -321 -224 110 21 69 27 27 -11 63 -35 58 21 67 127 373 336 -115 -553 110 442 -419 469 278 -250 76 107 -181 226 -85 73 -59 -293 424 -109 -43 -17 -74 369 -110 527 364]
96	3	2503.98	[473 822 146 -134 110 183 -18 -164 398 423 -142 -269 357 24 105 -210 -265 92 97 320 -130 259 -265 -220 -74 120 319 309 365 560 393 279 -516 59 -288 90 163 256 -30 316 -2 -84 214 189 28 127 78 36 214 -224 447 177 -270 -175 87 -280 -92 -73 30 435 124 19 73 -332 -3 -461 81 35 457 -186 45 144 283 292 -113 212 -34 548 29 313 119 178 176 -201 55 -412 52 -6 -114 30 -123 29 12 393 -499 41]

TABLE A.2: Solutions closed to record in the hall of frame of SVP challenge

SVP Instance		norm	Found vector
Dim.	Seed		
132	1	2929.33	[-115 -295 177 -35 -46 -600 68 463 -305 -125 168 209 36 199 143 -174 -133 105 -241 -331 165 192 264 110 78 -721 48 341 222 38 82 -180 480 -424 125 64 399 111 -79 115 -270 -122 -459 258 285 -6 7 -276 38 218 -389 5 54 4 -419 130 -278 -182 -119 10 -88 229 36 512 448 362 308 63 217 -90 110 68 -3 243 -139 -153 199 217 -37 28 -250 -35 99 -234 64 198 -139 89 -348 -319 -291 223 438 -44 299 409 -189 -41 153 -184 -344 524 -229 281 309 158 -569 396 -23 293 436 73 -506 -397 -43 -77 423 39 220 -382 -32 42 475 -41 226 -161 271 6 106 151 -338 -66]
132	2	2899.90	[-21 110 -58 245 40 -203 413 97 -545 -294 107 -126 199 260 -145 -243 -271 -161 14 -130 68 -18 192 33 254 -203 67 455 -95 -117 103 -184 223 294 -306 249 39 329 393 -115 129 303 -632 -602 112 -148 -177 133 -396 19 117 565 -161 -477 -341 22 -57 266 158 -378 -124 -164 -10 -154 240 -106 -138 -469 -61 -576 -41 321 292 -82 296 129 -310 336 -318 208 434 -52 570 -236 164 -196 -100 -540 337 383 -398 332 155 -78 -18 13 179 144 -248 165 -197 -278 -266 415 131 222 -306 -18 64 -15 463 253 -141 154 -95 -64 -42 7 88 -87 -162 -137 163 -222 -319 85 26 351 -70 4 -145 -291]
130	3	2968.73	[-173 175 -280 116 -61 -371 -224 -61 -120 52 160 173 311 -384 117 28 -265 -497 131 -143 -427 142 -567 -417 -5 -180 -63 20 -269 -552 -196 -379 -8 68 458 199 -281 -6 -88 -330 121 -560 -43 105 -665 80 228 389 -119 118 24 119 -166 -9 502 -5 -222 140 272 -324 261 159 -278 72 199 -117 -286 569 11 131 195 -317 358 65 -348 -142 70 183 -37 34 357 -88 -50 -440 336 -292 -67 128 -134 -233 -143 -427 137 628 -43 -116 -291 -118 -157 -245 -188 -190 173 -203 66 -426 -68 377 -145 -606 -214 -318 -454 85 95 -189 -24 0 -578 424 75 -13 222 142 34 -237 55 -76 -5 -79]
130	7	2914.22	[110 262 -221 0 -59 16 -177 67 -109 -80 -25 -73 777 255 186 182 243 178 285 -548 -94 49 131 -789 14 315 -53 -173 133 -158 357 190 -11 234 -353 -115 -19 -73 -272 -406 -67 18 -17 68 57 -424 -165 266 -292 -357 245 131 223 434 83 212 291 127 -30 149 -154 283 128 147 -251 148 121 -281 152 343 -211 -116 -27 244 -23 -57 -335 -311 -294 333 20 231 -35 -236 -55 -510 -388 226 243 -234 132 -78 -544 -99 -367 -523 1 -110 -484 -192 564 -84 145 -189 -26 348 128 456 409 298 -42 515 332 -442 109 142 -17 -145 -55 -90 158 -4 63 -254 -293 231 32 -75 -273 -188]
128	2	2947.45	[-124 -334 -4 42 -193 162 299 205 254 -18 191 56 408 248 29 14 17 -68 -99 165 398 94 -534 226 -391 102 403 327 250 -236 293 85 -37 -30 372 -109 -144 -88 -301 -106 351 95 103 327 -36 239 -683 -218 259 -149 238 12 -202 -173 157 3 252 -70 -287 330 -39 84 -295 -245 378 -144 -308 607 -109 -344 -386 187 17 39 -313 0 -156 -342 140 -713 85 -326 202 220 -251 -236 -136 -265 -82 -724 105 -395 58 324 -61 462 -3 226 -51 -227 183 -251 333 114 -467 273 218 -188 225 38 191 -583 37 -258 -404 242 140 -317 -177 -87 -87 69 207 -252 296 63 -355 -145]
127	0	2897.58	[513 -25 161 -131 -83 17 -475 -356 121 -412 -592 99 209 -378 540 -194 476 -300 471 -69 -213 209 -71 -101 -117 -83 315 -642 282 -272 -66 211 -341 302 -344 -40 -144 -242 -290 -23 -154 68 97 199 -225 -237 33 246 285 -89 -256 -514 127 117 153 348 -253 379 124 -84 12 140 545 -481 -443 57 -299 259 211 -484 3 -571 -483 -356 -89 232 53 38 68 228 180 34 -241 127 -16 -92 85 81 -404 64 116 -94 -173 100 94 -66 -85 121 -36 182 365 26 -71 168 255 -443 3 37 140 91 -402 273 203 -25 253 365 -130 35 -52 292 132 -70 -406 -229 -144 31 -76]

SVP Instance		norm	Found vector
Dim.	Seed		
124	0	2892.64	[135 -334 -246 -296 -462 495 421 72 -138 25 97 -124 -51 -37 -382 -455 523 5 -741 464 397 -416 171 -48 -90 -179 -61 -176 10 379 -447 160 258 445 251 -96 -67 185 -30 -21 -27 -232 270 26 -213 -317 -118 87 10 -9 262 493 -3 165 -269 215 180 55 -228 -180 267 -38 -91 -204 125 -161 -44 133 60 -109 -35 20 -6 440 -334 -16 -121 -2 261 149 15 56 538 -13 -629 27 -443 201 144 101 -173 66 -90 353 -313 146 -564 142 -446 -539 19 98 -121 -29 -268 315 -242 -120 513 -276 -200 210 298 -296 267 -322 -25 141 -107 -101 -181 145 -351 -87]
124	1	2872.38	[253 173 92 -97 225 -353 -284 203 45 70 -96 254 -237 248 -227 -112 123 -465 229 -164 234 142 -337 18 -211 -180 110 -11 96 -208 11 -107 107 -312 460 -96 -324 68 -143 257 -480 -489 368 263 184 -319 529 125 422 347 -399 377 -151 -296 154 -395 48 323 148 -214 -255 226 348 -153 -99 -490 367 164 341 351 -141 -218 504 72 -8 -303 422 332 28 -340 -64 -145 79 252 40 377 -45 345 21 -316 126 381 -81 350 264 -307 -57 -223 -340 520 -248 136 171 204 -472 87 58 97 309 264 -54 -253 -22 -55 -43 -28 -396 -270 104 223 18 -148 1 -446]
124	3	2886.65	[35 399 456 -479 94 229 108 22 199 -172 128 19 37 127 142 141 -194 -131 106 -67 -262 -173 55 239 481 -144 255 -122 18 -81 -132 129 -97 82 -682 -419 -58 227 56 209 -212 79 -72 -39 -291 -151 143 -89 147 -203 -80 377 -572 130 -285 53 214 45 127 -353 -310 354 10 204 609 -157 -37 186 3 186 226 -121 8 652 -368 -305 136 -231 -24 152 -339 27 103 -291 -280 130 -12 -667 166 395 86 167 -260 -479 383 -14 -470 393 -185 -183 -52 594 19 -235 44 289 -236 747 -591 -12 -151 -43 1 -102 -81 -37 69 -158 225 -226 -223 -99 73 -188]
124	4	2873.73	[53 43 -60 -596 -508 -488 198 -29 294 -385 -573 204 -287 27 -83 -409 415 -100 331 122 9 -540 176 112 27 -265 376 104 -63 107 276 -192 161 -73 303 -326 -150 -242 -40 235 -115 80 -474 -52 309 157 196 -246 -60 -234 -437 -56 -178 -350 -120 165 257 35 220 -216 94 33 138 -70 511 71 27 -200 17 313 27 255 8 282 205 62 352 3 -46 -321 188 281 151 -181 -281 475 -186 98 -44 214 393 187 128 185 375 -341 156 543 -21 -149 204 -659 363 -112 -311 142 256 -175 -375 58 382 207 195 13 -77 -226 -168 32 -148 -62 -199 -573 76 324]
120	0	2756.06	[158 305 -150 -24 198 -271 268 -277 185 -283 -312 -95 263 -31 -86 71 50 65 -209 547 139 226 -52 -33 -499 -68 515 164 -279 -16 -209 16 474 -518 138 101 -154 -276 -664 250 -249 586 12 345 150 67 -94 -306 168 366 -88 517 100 11 -560 -344 79 -102 271 66 -178 157 47 -184 334 54 100 218 -309 -293 -93 -195 280 -148 488 172 -223 39 -38 560 210 -99 -18 -44 -205 85 354 -132 -80 223 49 499 70 119 -68 -107 -406 205 -60 -163 -165 58 46 48 267 -300 218 112 -271 -148 464 101 175 -282 -116 178 74 -319 -326 212]
120	1	2792.47	[67 70 -151 -314 -14 -48 264 360 136 37 25 -110 318 249 -383 397 -439 95 318 15 43 8 432 -269 453 -106 4 80 571 134 230 -139 92 225 532 172 -32 16 3 96 -18 396 200 36 -101 173 -240 -95 36 -236 -136 -295 -224 145 -886 -101 368 30 287 1 568 -13 21 297 -178 -70 147 246 91 87 255 256 129 109 -90 259 178 -123 193 209 160 -87 188 -127 -554 -292 246 -31 192 -481 -27 -90 -413 -190 -54 30 -182 -233 -160 509 -161 -224 126 323 -10 86 -84 344 -306 -116 122 -97 -362 -169 -109 98 484 -130 594 -525]

SVP Instance		norm	Found vector
Dim.	Seed		
120	2	2778.82	[-149 -362 65 -30 -329 -40 174 92 247 -90 -227 152 234 -116 214 45 202 217 451 445 -166 565 -209 -324 179 -128 186 282 -144 -1 -222 -45 45 348 433 -336 -97 333 -25 82 152 -336 -309 -148 -129 -545 232 -387 81 377 186 -230 -259 -30 -405 -53 -117 -52 322 35 589 24 -155 -249 206 121 -113 213 -351 -261 132 -53 303 -185 118 -15 -174 -115 -341 150 478 -398 -29 -246 -121 -108 50 175 -6 29 -577 219 577 -356 -136 -54 161 322 167 110 42 -47 99 -387 357 424 -381 -250 151 224 -380 -7 171 294 -529 -89 -189 133 -209 49]
120	3	2842.70	[-327 -42 -570 129 -162 77 -118 204 -200 -434 39 218 180 -88 152 -146 75 148 28 -58 322 -357 -37 7 -63 -83 -414 335 183 -290 197 43 -271 208 -261 324 150 -258 -32 301 -9 -353 171 -481 46 -99 -258 59 337 -77 85 -80 -89 95 -585 85 -8 588 75 -381 -198 331 111 -91 218 -119 -37 86 -788 157 -479 182 -125 -51 -116 98 -224 145 -217 114 119 41 311 -252 -213 183 -329 68 67 334 188 -29 88 -30 -287 68 252 200 -314 639 -54 353 304 -462 207 25 202 204 -180 502 -710 -48 64 -159 -365 584 -16 -86 228 265]
120	4	2729.25	[214 129 737 112 -285 179 -117 1 -21 275 -41 -490 -447 132 144 186 305 -4 -743 714 99 -46 264 -242 -48 -265 2 78 -132 185 -508 554 -80 -179 -82 322 151 -38 16 -99 8 43 89 -357 -12 -76 -275 -65 -418 -131 -295 -44 36 -199 -192 -482 419 -279 -149 -30 71 -149 244 467 286 -266 -335 67 99 -229 -90 6 155 102 128 -76 -298 -83 -158 92 284 106 -314 -158 -124 122 -67 331 94 -166 -474 -219 43 -651 82 -156 -512 131 -18 218 -235 -10 -93 29 -38 88 -82 87 -56 195 10 -246 327 200 -26 28 76 -460 -3 -103]
118	2	2789.09	[-246 -87 -90 309 105 235 193 37 -52 -301 -356 -274 211 -415 154 -26 -408 255 -199 95 107 -173 -48 17 63 287 316 -400 -255 119 101 5 58 -257 107 140 -124 -81 -18 5 66 328 158 638 -286 -404 487 -436 449 429 167 201 -415 -59 530 125 252 99 63 -65 47 218 17 -111 730 -158 188 -229 -22 -572 303 -101 -280 -377 -268 -326 -285 -126 373 -348 -244 -262 300 309 24 -56 -361 -444 145 -42 -21 -7 -149 -155 28 -243 -235 -179 123 -15 -16 -380 530 -117 -80 -79 260 -271 110 263 163 311 108 210 286 -23 183 347]
118	3	2785.57	[392 370 -189 192 30 159 -64 -65 -31 -150 -512 787 -126 -41 164 -155 -252 -428 31 263 -112 150 -223 304 -505 72 -389 94 -12 293 579 42 -249 155 -255 83 -216 -175 73 2 323 488 -2 203 -111 168 -51 126 155 -463 -507 111 -44 -465 567 -103 221 87 -359 380 -12 -541 -107 -339 284 198 271 71 -187 109 131 124 116 37 -326 165 185 288 -253 10 201 17 -209 -295 148 -182 -308 -535 -402 39 -455 -139 18 -105 102 322 -81 103 -283 255 -188 103 277 74 -114 44 67 -80 -522 -29 -198 244 278 -87 226 -44 -35 -60]
118	4	2793.29	[-85 117 -153 -22 -351 -175 -149 -166 -157 -198 -50 -466 -31 261 149 8 -186 -157 -301 -139 113 -93 -485 127 -296 84 -552 -359 274 285 396 488 -623 171 -19 -130 210 -148 223 246 96 290 -14 87 7 -15 226 188 -45 45 340 -304 -359 204 46 -405 427 427 -207 92 10 -15 -23 -295 -49 53 -319 -443 -147 -208 113 130 216 202 330 484 112 -204 -300 9 -240 -221 215 3 -183 106 17 226 523 -58 45 -288 -255 -309 -95 -521 255 284 -653 468 -193 317 14 -231 67 481 -387 -98 -25 -240 -172 54 61 198 -217 -77 -341 319]

SVP Instance		norm	Found vector
Dim.	Seed		
118	5	2764.17	[-112 117 58 -320 150 -163 81 219 -221 -213 326 -301 -637 -580 -181 -235 124 -136 69 -166 708 118 -7 119 -170 316 228 294 -115 179 102 162 161 -104 -17 -344 -481 -372 294 251 -362 -326 -396 -46 172 -30 39 98 -169 216 -366 -124 -379 27 -270 -349 -336 -294 -248 67 -22 -137 -11 42 -372 -98 83 285 -115 113 200 370 304 -271 -177 -242 -64 109 415 350 255 -365 257 258 -374 -211 200 -27 227 -220 -168 63 258 361 -108 136 -407 63 85 -345 -207 -234 11 -262 29 294 -119 -165 -545 210 228 55 86 -462 370 -270 -160 -113]
118	6	2768.58	[-120 430 144 40 260 254 -153 -275 -388 325 630 -42 -107 614 61 250 302 -262 -106 -428 -200 -538 72 -28 -7 -482 -45 -238 422 -98 -251 307 20 137 169 -29 385 -231 461 -85 -178 -94 -363 -259 -124 -323 72 330 -166 -187 -94 -64 -113 147 282 80 95 -267 -261 -79 -220 52 143 -5 -392 -302 320 19 174 47 -38 213 -269 230 236 13 97 -364 -181 50 -122 21 -86 -455 440 16 706 -55 -331 47 387 98 315 298 -74 -12 -81 -309 2 79 -235 -149 -212 -106 181 -354 53 241 -344 -237 -75 -83 135 348 -76 1 -254 616]

Appendix B

Lattice basis of numerical experiments

B.1 Well-reduced lattice basis in Figure 5.17

CMAP-DeepBKZ has succeeded in finding a sufficiently reduced basis, as shown in Section 5.3. Here, we show the part of the basis used to create Figure 5.17.

TABLE B.1: 57 lattice vectors from the beginning in the reduced lattice basis of Figure 5.17

[[110 262 -221 0 -59 16 -177 67 -109 -80 -25 -73 777 255 186 182 243 178 285 -548 -94 49 131 -789 14 315 -53 -173 133 -158 357 190 -11 234 -353 -115 -19 -73 -272 -406 -67 18 -17 68 57 -424 -165 266 -292 -357 245 131 223 434 83 212 291 127 -30 149 -154 283 128 147 -251 148 121 -281 152 343 -211 -116 -27 244 -23 -57 -335 -311 -294 333 20 231 -35 -236 -55 -510 -388 226 243 -234 132 -78 -544 -99 -367 -523 1 -110 -484 -192 564 -84 145 -189 -26 348 128 456 409 298 -42 515 332 -442 109 142 -17 -145 -55 -90 158 -4 63 -254 -293 231 32 -75 -273 -188] [178 199 -253 123 517 180 -42 -315 -830 -20 60 -718 340 58 -173 204 -52 265 136 -325 -115 -234 99 -471 320 101 -406 -211 67 -85 211 75 155 397 -628 -87 119 91 214 -723 -18 -352 -191 -30 -172 -458 -276 -184 76 234 -112 -82 121 83 130 389 -65 117 -104 135 243 160 -51 352 346 -237 34 -77 86 32 -137 -278 -375 201 559 -135 1 70 349 319 -157 340 -16 369 -259 -26 -652 -108 -137 -313 -78 228 -105 414 -136 -230 233 -321 -236 -290 255 -265 337 -150 -381 113 76 40 95 290 -57 113 153 -751 -466 -249 -5 3 114 36 423 522 109 96 -338 277 228 258 307 -94] [-136 -2 -90 494 313 228 -17 -35 -435 -164 22 -308 -53 308 -114 52 294 82 -105 -17 26 479 107 76 4 -219 -4 -460 275 -18 135 -163 279 148 -135 196 228 -92 393 -118 -14 388 172 311 -89 85 257 -69 90 -84 300 -385 -9 190 -117 115 116 -376 46 -110 454 -148 -276 -53 494 267 -479 -42 -32 65 46 82 420 -123 17 373 351 -40 285 -261 -137 188 640 650 -166 245 -202 -433 -105 1 -750 727 -540 -240 199 467 277 -118 -122 -300 -62 -229 208 -227 -297 296 -125 129 -57 -141 71 -117 -100 -373 -228 27 230 -586 106 48 -11 -219 -33 137 -98 -93 -11 -63 460 -221] [132 -19 811 -130 -293 189 373 -240 159 558 -10 -24 -273 -436 -336 -31 -284 108 -250 286 260 140 269 152 227 44 -126 -244 -180 161 65 66 -350 -136 72 -51 348 -16 100 -44 241 -47 53 336 -340 51 -52 -724 -515 433 -39 180 -69 -177 53 189 331 -142 141 -46 363 -78 3 268 -357 -183 -513 369 -58 21 222 220 -769 -242 412 -196 -148 206 -105 -199 -447 -37 -164 -43 -137 -105 11 -61 -349 96 239 173 117 10 -251 321 -326 19 377 152 -578 -150 -35 26 -186 -404 538 -573 -205 59 -199 62 177 240 -244 -44 -218 223 15 -169 11 398 -151 230 -467 208 103 594 120 305] [-103 143 292 5 450 47 143 174 17 198 233 -412 -136 178 -299 198 -369 -233 -21 -258 55 -68 -351 -478 -122 -304 240 -73 230 -237 -554 79 26 -76 138 -225 57 -161 -59 52 441 -332 -486 -18 -102 -408 -124 -604 -276 309 653 314 49 77 -19 -56 356 -26 19 389 93 -115 145 267 -572 10 60 517 -269 260 -289 -11 -334 -88 154 -149 -81 -164 218 -287 -48 -99 -2 158 329 -20 308 -121 147 78 683 -251 -47 -242 -56 113 78 -65 59 546 -166 180 257 307 77 -93 585 -353 -79 595 -32 66 304 -420 -60 -9 -148 -362 -202 189 -150 549 245 203 -390 479 343 349 -65 214] [-265 316 -96 -20 -613 126 -329 -463 56 108 -90 193 335 -423 -183 -587 466 -29 47 -238 207 310 151 441 -376 -43 -6 -535 -342 263 449 -244 286 -18 219 -42 298 115 -122 -213 274 79 182 319 101 311 -62 -12 -442 -338 59 309 106 340 -89 342 117 -586 565 256 139 -201 216 -161 -120 596 -442 -176 -23 219 -163 66 -27 -23 -259 443 -115 89 -3 234 171 -317 -133 -138 -306 59 -131 -147 321 -236 -338 231 -155 -543 7 88 -493 33 -53 -199 413 -227 -116 -37 160 5 26 275 416 -279 -471 -44 339 385 4 -154 -37 413 261 269 212 -480 -168 -556 -234 309 31 -109 -372 8] [-126 78 -8 225 179 153 202 -609 -604 351 -41 -216 452 91 36 320 -351 -123 9 -519 -40 -122 -468 -497 -200 -100 -72 -559 -62 -474 118 -286 16 507 -614 -478 -21 694 660 -355 -5 -521 0 -224 -443 -514 -539 127 -41 45 238 312 -61 104 -230 303 -188 -442 -231 -150 -277 229 567 93 3 510 -107 -111 -104 555 112 -312 -132 148 550 -67 288 -337 382 -266 25 54 -48 170 66 -263 -187 176 -156 -456 284 122 -52 -189 -267 -363 334 -244 211 -174 336 122 190 158 -257 -49 459 101 213 117 188 -150 26 -625 66 -30 50 230 -260 292 31 135 125 -17 -208 159 -202 -13 224 -111] [-176 283 -155 -30 -108 -221 -160 -358 -361 -7 368 -342 330 -155 -104 205 263 -185 -140 -158 -18 -433 82 -208 12 -304 -108 -32 -349 -268 78 46 -135 544 -536 -595 199 520 340 -16 -66 -86 -140 -116 -9 -15 -605 473 156 91 -94 327 -145 -118 -121 38 -605 -223 247 -143 -291 -43 732 -340 -368 586 58 -109 -389 -130 218 20 -109 27 -173 298 439 -355 -30 -191 87 -26 265 48 -130 -130 285 128 -86 -543 23 32 277 267 24 -89 -96 -58 -287 -291 -59 172 -641 203 -114 168 79 -222 57 157 -123 262 72 348 319 -474 3 566 487 58 40 -420 473 274 228 -217 170 -99 -244 -452] [-70 -23 167 -16 -391 60 -91 -279 -152 380 -158 910 628 -305 -140 -255 79 -174 4 -313 145 275 -451 -217 -339 280 410 -357 -448 -388 197 242 396 -157 281 -302 -333 464 -775 -222 188 -170 175 -124 -175 -142 255 -212 -201 -59 307 690 427 434 -116 -121 700 -134 342 -70 -55 -236 496 404 -528 356 -289 -121 -31 540 -685 -177 -63 94 54 -8 -874 -195 -344 82 46 -153 -214 -742 125 -110 149 158 320 286 705 -163 -205 -556 -161 -165 -393 -165 333 502 203 117 -58 112 775 -144 217 483 405 -72 94 10 -86 385 -411 -168 -488 -44 -213 -218 119 581 -13 -894 -379 244 229 416 -534 88] [-362 -554 -277 590 198 202 -138 -442 -645 262 -672 452 -101 -442 -188 11 -216 -200 182 -10 276 -44 -131 -323 -168 -601 116 -586 -80 -602 89 -308 452 11 204 243 223 402 257 52 -334 -32 390 -194 -298 -26 -258 -326 11 -50 290 168 47 7 -219 -145 178 -364 278 106 -293 152 -224 -92 337 113 -586 155 -4 576 -146 -322 457 -170 526 -283 239 158 647 -416 194 -554 -59 253 13 -54 -40 -197 -205 324 350 400 -84 -297 -300 -36 40 -511 356 -296 208 -118 410 78 -185 -414 31 95 -143 -450 78 -620 -570 58 -621 -121 1 -260 186 -361 94 9 -347 -516 -110 540 -339 341 655 2] [-84 -322 7 -5 -237 186 -195 52 134 372 -357 71 -586 -573 -377 -511 48 -484 190 -146 27 -564 164 854 -25 -105 -36 -196 268 486 89 63 273 -361 244 410 -36 -389 -626 77 298 -152 241 195 -135 30 145 -145 -143 -24 -439 -62 -366 -215 219 -107 124 14 -150 137 277 -98 -662 -42 178 -186 -245 -259 -142 -307 108 -195 -219 -76 375 -281 -150 412 182 -57 399 -317 -96 -574 -353 -162 -586 -105 341 493 -349 -97 221 -203 419 214 -491 -272 259 -32 -185 -506 327 115 -567 -292 -440 82 68 323 -768 -464 6 436 -376 -99 -403 -87 245 47 -113 -159 -373 -66 -127 392 -119 -96 79 262] [-12 426 38 -53 224 -176 -571 -796 -651 -326 177 -369 -33 -213 62 -22 -109 -17 342 209 -147 -119 -194 590 -179 63 -497 234 -439 260 -39 -249 -473 -246 -57 -404 -283 620 492 -286 196 -29 158 -112 24 -123 -458 328 175 286 -678 -407 -165 -52 -391 257 -725 -368 -255 8 -29 30 511 -573 569 -60 -156 212 507 -51 214 53 -157 -46 -256 611 422 18 -65 -19 95 171 -177 483 -31 442 147 158 -5 -716 -180 51 150 343 510 493 -93 242 147 -318 -199 155 -102 30 56 -62 -106 -145 167 -687 191 7 343 66 57 -202 351 213 -387 929 155 162 866 196 506 -616 -154 -325 -131 161] [174 -185 32 411 -93 228 -40 114 -602 -93 71 -460 26 -89 -696 -173 512 -470 73 -325 -158 -534 59 -484 -70 -508 374 -590 117 -221 -99 -91 458 486 -587 410 -36 -13 103 495 35 -533 -196 -180 4 180 -1 29 -155 25 124 150 -125 124 -475 -187 123 -223 16 160 -168 -329 100 61 12 399 -155 491 -265 -294 2 288 120 -487 623 -311 539 -272 504 -428 27 -235 105 -154 -281 -59 180 -77 233 530 -106 12 48 53 -56 176 -153 -216 384 160 265 -48 -116 365 -428 293 -140 -165 334 462 -124 198 -428 387 -325 -650 -617 -431 486 -575 -65 -13 140 395 -49 48 357 -9 -118 -464] [35 -117 311 -30 591 -314 -86 -6 392 82 -57 197 -284 45 -123 70 -597 127 -181 190 256 -215 -174 -326 -16 -39 181 1106 -154 -227 -576 252 -367 -361 1015 -452 -212 -85 -370 -253 587 139 -186 -40 94 -81 -325 -772 191 546 -46 -112 -111 -274 309 -474 329 159 -60 686 102 -175 -139 -140 -441 -718 347 759 371 329 -712 -95 -321 -31 -388 48 -397 498 -289 -144 -254 -131 284 486 374 -250 316 -135 -70 -38 1318 -74 177 766 -470 235 -232 -215 325 114 -527 56 150 -350 764 -482 -142 -204 -543 -594 207 -106 -312 601 -230 100 -52 -413 134 -84 60 655 150 350 -74 -60 183 366 403 209] [237 177 369 -76 -529 308 202 -342 343 34 126 -971 -145 254 -749 -594 338 493 -44 -641 127 -113 654 24 288 661 292 137 7 473 303 19 -101 355 -322 -136 472 62 179 -520 150 -159 349 678 154 440 193 -137 -305 128 -826 -97 -62 -27 153 763 -433 -133 51 31 201 -391 -70 243 -514 130 200 13 -46 -400 121 223 -762 -409 -410 84 201 290 -147 531 -344 376 438 217 -570 246 -438 -451 -272 -78 -989 103 516 215 -14 198 -665 235 40 -400 179 -309 163 16 -389 329 54 -140 401 528 -276 391 -88 153 105 -26 250 268 301 -19 100 -679 -180 353 -200 -176 408 -306 -299 17] [38 -226 4 326 -233 -81 500 167 197 162 -369 1023 198 -142 96 -60 -163 540 -153 -80 368 -315 308 -768 160 457 208 71 118 -266 -78 182 114 61 670 -121 446 64 -245 -543 -175 486 35 124 273 194 386 -282 334 -52 246 -303 409 179 539 -145 894 390 3 -411 122 -102 -450 534 -101 -396 117 -211 337 508 -454 138 88 113 -256 -93 -454 658 -617 541 -416 -104 482 53 275 -218 -12 -433 -523 291 155 384 -540 194 -548 -18 102 -70 -6 -301 25 115 -116 -372 325 401 26 575 -308 -695 -74 116 -429 -391 226 901 -159 -349 341 -912 103 -773 -762 -292 -419 60 -465 505 302 -325] [24 289 -1016 146 353 -537 223 410 341 -262 184 -72 86 -338 718 120 -239 254 -480 363 16 -452 -90 -276 450 -403 -508 445 182 54 -189 55 -317 263 -180 227 206 -373 326 608 -526 -163 -442 -355 577 -79 -502 26 -24 57 111 -145 60 8 441 93 25 397 81 -142 179 227 -261 -145 521 -280 392 -76 -621 -237 173 254 -84 342 202 -404 189 -67 87 -102 45 -133 -251 59 234 -253 48 407 122 308 172 120 290 -84 -15 -658 941 -42 -136 90 107 490 -285 62 -26 251 124 105 -307 106 -124 -312 319 -407 421 210 -29 358 -341 -561 -272 -696 -132 31 71 -58 105 288 459 360] [-428 -92 -123 125 -169 -338 83 162 306 234 380 319 -9 -318 10 -125 -68 -191 124 -105 800 -843 268 -339 97 -279 -264 682 222 -423 -37 -268 -115 64 14 -238 133 77 153 136 -154 -320 206 -271 68 298 -70 161 -32 120 -156 1 -117 -427 275 -311 -95 -69 517 -227 -290 -106 61 376 85 124 -39 -254 -670 -159 -340 -103 196 -367 -207 -493 1 114 -306 390 -83 -82 727 -26 -266 -235 -107 -95 -532 -46 576 90 398 178 -108 140 -230 42 -9 -382 -757 305 -194 -309 334 432 -120 81 -72 -32 -423 -104 -261 517 180 -182 -40 -163 580 -638 -345 -692 -199 56 90 -476 -19 36 -12 -88] [425 98 -8 111 -324 525 -137 -831 -170 -289 -323 -68 -391 -436 -548 -325 36 483 701 -419 -120 162 451 -498 414 23 -10 -104 -308 399 348 491 243 -285 -38 732 385 632 -144 -319 -115 187 512 210 184 168 -341 -315 -403 197 -295 -161 -63 -37 -283 461 -249 325 -335 245 215 659 -330 8 -234 -576 -273 256 791 -3 173 27 -426 204 484 -665 -321 -35 -41 -150 504 -7 -793 -111 -107 120 -714 -8 235 122 -324 -38 -58 -321 -252 -391 -327 -332 -104 -646 723 -240 0 178 -510 -450 516 191 453 172 -60 -214 232 -421 -852 365 111 -44 -415 -277 129 321 -63 -508 -124 488 337 292 -24 470]

[-357 -229 -216 195 -118 -181 -683 -294 475 -310 -853 -5 -418 -283 -91 -76 185 -135 562 79 278 -521 796 93 -253 359 255 218 -220 393 -434 -94 -115 -619 965 -279 661 166 -238 -81 -236 328 -453 103 251 337 -269 383 -164 -410 -341 -119 341 -226 -240 -86 -357 315 317 170 -594 97 -277 -450 -452 190 141 166 357 -293 -68 452 160 -390 -940 570 201 517 51 113 -251 -127 -276 375 -48 282 760 -141 -200 -11 29 -106 74 797 89 455 -909 229 -419 -210 -7 -19 397 -241 102 318 -480 -71 -267 -272 -103 491 111 832 311 151 22 364 510 -322 -130 -1096 -180 -133 138 493 -195 -462 -562 127] [-90 -595 448 -348 -263 513 140 -11 640 72 -707 105 -640 170 48 -283 -681 -604 -203 141 -223 325 70 1021 -299 108 106 -175 227 526 -272 -122 -108 -273 413 -285 -135 713 -454 -9 1095 -474 -41 470 -357 -389 187 18 109 109 113 150 -665 -622 282 1 -154 -373 -558 -336 453 407 172 113 122 314 65 -74 209 -97 462 -4 -7 -89 46 169 -85 50 -48 -518 114 -136 -262 -161 19 -60 46 240 -214 -341 -153 -168 -161 -213 137 61 -153 83 303 325 -149 -27 197 395 -153 -33 552 -332 86 253 257 -374 531 135 345 559 16 386 -291 491 -209 200 -207 280 188 64 -313 -136 -408 479] [-122 -385 104 78 61 45 -631 29 -616 -448 -536 88 -322 55 53 -254 -303 -453 189 650 -405 8 -302 549 -768 -287 154 236 124 -29 -514 -484 -288 -567 566 185 -467 -283 406 565 -133 -76 325 -557 484 -216 392 -66 235 -184 -67 -157 -305 -26 -88 -119 -293 -235 -68 460 -323 343 -300 -287 262 -163 647 553 418 150 320 -31 170 -67 -351 -347 438 143 126 564 413 -589 -395 278 4 271 168 30 231 -16 89 -877 163 -525 452 159 -443 52 240 247 899 326 124 823 -216 -211 229 -410 -132 -401 234 -415 -296 -579 27 138 -146 118 -203 693 390 235 104 41 362 -126 -460 30 -196 280] [-247 43 494 348 -72 11 -156 526 329 -16 157 91 -243 29 -355 -524 -77 -431 529 -642 361 -386 36 -698 -234 -292 631 68 554 -298 -183 -275 -251 -353 218 468 -42 358 -494 321 733 -286 820 253 150 410 -89 129 -612 82 765 -826 -350 -41 163 -143 234 -695 -164 642 -231 267 -105 143 6 -64 124 331 522 471 -401 -200 146 107 -510 -374 278 54 -124 21 412 -466 248 -153 51 -329 90 -360 -177 565 233 113 -826 -209 -71 -30 -88 52 197 20 -84 108 388 -174 -149 274 99 -205 513 149 -426 -505 49 -104 54 441 95 -1618 -1 -78 -487 -248 28 232 117 -136 -178 148 41 -238] [307 -209 -552 -42 637 -295 -161 665 193 507 6 -115 126 -446 200 -204 -18 -141 -111 -125 90 -955 466 -232 42 185 -6 105 589 -146 -143 -202 33 389 16 -87 -65 -292 -793 -111 274 -580 -83 -181 435 -307 -342 -55 216 -60 1 -252 -165 -302 636 -376 714 -266 167 374 -102 21 -474 605 347 -55 719 610 -266 -302 -332 -169 -92 155 -163 -701 96 975 234 388 5 -542 418 -530 67 -588 -136 -270 -259 575 402 38 -23 891 -673 -489 254 -100 -125 287 -11 -290 229 -157 136 620 -574 -290 121 82 -409 -483 238 -331 -222 222 -269 -463 244 -524 -34 -268 -371 397 -72 623 389 466 -33 -218] [73 192 242 -32 42 -141 583 -159 419 452 539 258 -508 -81 184 174 122 146 -328 324 155 7 233 208 186 -131 -271 172 -251 120 -207 -181 -94 300 251 -86 590 306 865 78 -163 271 170 -240 344 618 -184 -179 31 -47 -391 -223 -393 -205 -100 96 -319 -88 -211 -253 -51 -178 36 5 -255 -156 158 309 -220 -377 624 719 -114 351 -207 223 464 210 -523 -199 25 -450 554 8 425 24 -145 -66 -372 -677 -463 473 448 -44 -281 382 -58 572 -26 -919 -25 51 -780 239 -257 -277 419 -243 98 -669 -445 -505 128 -151 -207 487 598 -95 -168 -231 184 -1006 32 494 64 -312 -353 361 911 -517] [-29 101 -446 -21 64 -31 -307 -682 26 308 -107 208 324 -705 -261 -315 -358 -241 641 -121 413 -683 -331 -201 148 -502 -809 -76 -255 -156 -16 -237 111 295 -691 45 -376 200 97 24 -196 -170 75 177 -117 -203 -625 -332 -202 353 -240 213 -155 -149 342 12 324 -158 -63 338 178 -235 -173 -171 840 -451 -851 -510 -211 295 -356 -580 -271 53 745 -101 -410 73 69 -62 591 251 -132 -33 -153 -2 -344 135 257 11 798 -48 292 -466 245 222 421 -548 790 -536 -182 26 -190 51 -407 -268 -458 691 467 -441 -328 -342 -49 345 79 -314 -161 141 -92 448 51 374 157 -215 -445 -287 -214 -668 791 472] [-303 -337 -223 -301 281 471 -304 9 -938 131 -71 -547 -56 82 -63 105 -339 110 -433 785 46 257 570 162 200 -143 -179 -357 196 -157 267 -367 88 -60 -127 -329 388 103 105 -430 50 370 433 13 -64 -289 -27 102 543 307 -427 453 -83 -150 -146 294 -274 -45 433 67 261 -568 319 450 -265 300 -351 133 -384 -445 238 -412 219 -501 -137 232 424 313 104 -304 -287 -365 810 -99 3 402 -366 -200 -332 -491 -197 379 218 306 343 190 -458 523 -870 -442 -660 -239 274 73 -417 -116 355 -572 -71 230 -149 57 532 -206 -564 -154 632 -210 419 -99 413 -161 216 -231 -47 657 275 648 -80 -344] [149 -136 161 622 -507 -92 -260 -137 -413 285 -40 562 -89 -1110 -119 -438 190 -277 -103 424 -74 -324 -315 587 159 -464 -636 -353 -386 46 770 358 -816 -61 136 496 64 382 -324 519 115 -316 515 -399 -208 745 360 -197 -679 420 -188 -69 -101 409 42 248 -108 -621 -202 198 270 103 332 -586 460 -249 -449 137 344 38 646 129 -295 76 610 -387 -47 -597 176 -66 383 -593 -721 -826 -350 -645 -112 412 149 440 329 -5 322 -549 164 -203 -219 -62 589 192 -257 -45 -454 -77 175 -452 178 72 58 -415 -359 -265 -190 564 -410 -732 -360 8 87 -190 328 546 213 -758 368 -160 158 738 -198 335] [-28 -356 -177 -251 -250 323 528 -25 -29 330 -109 -54 458 -109 97 353 338 192 -544 -300 -296 32 435 -623 468 486 10 -576 271 -648 828 708 206 536 -309 -221 614 -150 113 -269 -422 399 552 182 -559 118 -113 -45 -139 19 -155 624 572 693 29 795 92 400 -82 -378 276 -664 456 80 -631 -71 -614 -429 -24 185 813 74 -63 -454 317 448 -42 -503 -521 -285 -306 173 533 -778 17 -537 -145 223 -190 474 -237 451 316 354 -239 -544 118 -3 215 -130 -195 -54 -439 187 -743 -288 360 500 -13 364 -92 599 -251 194 638 -142 99 639 292 -769 -3 -724 137 -159 -87 287 -47 392 -122 -253] [-194 37 -616 -155 68 169 -249 -188 -633 -159 237 -211 35 311 -89 26 -112 291 -470 447 187 -137 -181 -65 -100 -213 198 -203 -324 -621 332 -568 276 445 -397 -21 42 1 512 -188 -648 402 537 76 -289 330 -442 673 471 -288 -484 -52 -103 -240 -317 -548 -311 -403 323 -176 -938 -235 186 -355 53 295 -257 -129 -428 204 -285 -915 296 64 83 -34 682 307 280 -469 40 -620 513 32 -260 117 -443 -368 -255 465 -574 574 190 281 64 -334 282 -568 -168 -350 -5 -470 75 -394 -742 5 -585 -218 -246 41 194 -59 -20 -53 -355 -166 297 -102 -111 33 -347 -572 -14 -398 10 500 -238 -207 256 -223] [-55 -65 -811 64 264 -317 -199 59 -1032 -649 -128 126 -308 -381 -622 -159 -522 173 188 -266 -244 -994 372 -437 755 -365 -84 373 -457 447 -381 238 221 -266 238 319 -62 528 -71 -203 -76 -2 -677 173 96 -573 -182 25 678 897 -1003 -168 216 39 -189 -128 -176 864 45 -505 -250 -612 -100 -477 -60 -465 -115 -87 315 -182 -421 171 -165 -630 345 -138 81 578 328 -557 42 92 -685 370 156 463 244 170 443 212 -126 -49 750 739 306 -77 -143 585 -56 9 -303 -19 337 419 553 -382 -326 232 -532 35 60 496 -506 358 -292 -461 -239 291 859 -675 195 -360 351 69 579 60 -381 347 109 -438] [-127 58 917 734 -635 -483 462 -128 -299 242 -344 354 435 -56 -418 37 523 176 -163 -794 9 -131 -199 -81 -622 -376 1310 -435 188 -177 202 -236 -382 365 224 -766 273 164 -427 -69 1006 -430 -124 556 193 121 64 -194 267 89 560 50 -189 -25 -534 317 218 -475 331 -225 -174 -210 468 -283 -101 789 67 714 -61 559 -79 242 163 -462 -686 153 370 -302 280 -157 -138 -463 362 237 147 -279 1094 -170 -253 387 -305 -93 -556 -505 -382 291 -131 1 85 495 -427 242 0 522 452 16 90 -422 50 -12 93 215 -527 523 258 -424 -424 82 563 83 -266 -69 -256 395 482 -233 -79 85 -671 -286] [-59 214 -365 -358 280 -118 479 -378 599 229 211 -1150 -430 -312 187 -10 457 743 -200 -157 -98 415 508 161 387 334 -212 397 -324 -285 -75 560 288 315 -644 -488 254 -428 -480 -420 -547 367 -307 536 -333 -916 -417 -693 145 140 -423 682 453 -193 686 -99 -102 387 420 -245 145 -259 168 -275 -862 -288 365 -78 -731 -541 52 -6 -300 -126 -178 179 105 261 320 -24 -538 320 -197 169 466 408 412 -285 99 -52 -229 209 575 366 -450 57 551 -519 -144 270 -170 103 -174 593 458 -22 266 -670 -146 250 385 223 -463 240 532 252 716 1188 -391 481 101 -275 100 -13 -110 323 687 -270 -72 567] [-14 159 446 -466 -335 -151 79 -169 -564 -16 561 497 1006 -84 74 -123 471 256 -614 29 198 -454 45 -633 -43 775 199 560 -576 -62 249 -137 -543 284 -18 -627 -4 353 -341 -317 -32 45 154 -390 485 279 -413 670 -485 150 -698 -591 36 145 -48 -268 224 72 175 -120 -903 -368 1040 427 -297 61 443 217 -97 231 -291 84 -187 -30 -175 -259 131 161 -1026 235 -376 -185 -53 -544 -49 -300 26 309 -140 333 344 49 -17 452 -454 -759 -362 336 -210 178 -351 -141 -385 -277 233 635 87 192 528 -433 -130 605 21 311 -132 88 -424 420 -335 -714 -316 -371 233 -50 -342 -618 -523 519 -500 -340] [248 701 -193 300 -193 -696 -289 107 -579 -105 361 42 383 -83 -277 635 735 42 144 -372 -56 -470 41 -312 -235 -688 752 -818 -104 -473 -398 230 806 242 11 54 325 -752 -159 116 -206 228 -731 -435 506 95 259 99 550 -298 783 32 289 157 -338 -505 629 275 306 -307 7 114 -51 431 -116 247 731 201 -494 91 -743 -541 -50 618 -305 -430 95 -77 -21 304 831 -277 272 -103 110 329 -218 -889 -19 -73 -696 -546 -782 -224 322 312 253 -436 -689 -66 555 -28 -676 387 -506 553 -689 -385 -117 990 -553 -60 105 -717 -336 -137 -756 -147 596 166 92 241 132 305 -150 162 550 -7 -177 -995] [-876 230 -637 227 444 -618 -116 128 -509 214 -329 478 150 -362 -44 -482 -162 -94 -813 -116 -399 -263 -566 -143 -692 -472 747 375 553 301 -26 -255 118 -526 279 -408 -406 -381 -248 100 307 -234 -381 -238 516 -115 -689 203 103 -148 411 -205 -8 122 87 101 -33 160 617 1094 183 -326 -462 -238 -35 484 458 163 -178 606 -137 -269 81 32 -368 198 229 328 811 184 -33 -542 142 -65 100 -240 327 515 193 759 1013 -157 -298 27 568 -731 -33 -619 -387 613 64 73 525 131 272 -319 -554 389 -312 -26 -545 -717 -336 221 118 -257 -1031 308 718 -87 -50 -56 -908 211 -435 481 606 182 50 -50] [199 47 -74 50 -225 35 321 -655 30 -175 0 -57 636 -92 525 -150 603 -27 245 -302 428 376 51 145 -33 401 -271 -246 -172 -286 606 -864 -204 41 -460 126 294 -53 366 279 -42 -400 940 -469 177 500 -295 -386 -579 -61 -348 13 -265 -100 -168 408 144 -359 3 98 -415 452 361 439 614 360 80 322 -152 -221 182 -184 167 277 198 -360 399 -173 -181 113 231 -158 237 -193 -389 -111 -585 -305 7 -148 14 31 94 -363 -277 -393 25 111 528 -499 221 -147 -62 -228 -648 -53 373 -244 756 -13 -258 -302 626 -646 -416 128 17 -201 -903 -37 -46 97 -64 139 -342 -260 218 252 334 602] [-299 -277 -218 325 -273 -150 294 987 486 -401 73 -95 -951 -112 216 -10 185 -429 -921 872 -582 270 369 997 -76 -425 45 114 -4 301 -45 372 -329 -1 1078 132 576 -350 -167 280 472 84 152 565 146 582 396 119 -334 -483 414 -348 -449 -164 364 254 -250 -685 -115 234 702 199 -482 -843 56 -258 -33 -81 21 -340 808 469 -400 247 144 145 30 -696 -29 -2 115 -745 181 -176 -111 -528 -87 529 -38 576 -962 349 114 -597 301 -42 245 75 -20 598 -143 -309 -579 373 -249 -149 273 -90 -301 357 -622 -126 371 119 -69 -757 -253 324 38 -267 -74 -857 36 142 286 349 63 499 280 391]

[377 357 189 -300 128 -147 -284 462 -135 591 187 269 661 250 132 -164 906 205 -475 307 217 869 -133 1103 365 421 -373 -774 -253 1224 673 133 -305 95 314 262 401 -308 -1 -3 -5 851 -34 402 167 321 281 -121 -501 -131 -361 -181 -17 957 300 -232 712 -304 419 297 -73 -151 51 -310 -233 555 224 -644 -140 234 -185 -165 -202 820 -589 587 150 670 -294 92 510 13 -12 -27 -311 51 -493 -291 304 -627 -439 88 -42 -280 247 -19 -750 146 -722 79 -23 -1121 120 -553 -652 -129 -687 -357 -84 -207 -395 162 522 -339 -30 80 215 251 146 621 363 -104 -35 -40 -366 65 -158 279 540 -387] [825 241 113 428 49 -136 -111 -16 -355 307 84 150 -8 -113 838 522 -338 42 680 -483 123 -1139 -482 -244 -659 114 -286 -4 136 -102 -549 -258 -380 -70 130 -331 -106 -437 80 -1159 122 -348 -544 -716 -477 -919 -275 28 214 -375 400 -5 241 -145 -741 -607 563 -214 -612 -353 73 331 -216 62 324 -640 -107 -137 658 50 -659 -100 -122 25 1033 -747 -820 -124 20 602 -272 529 -347 210 403 -652 27 365 194 -456 705 -400 -335 459 -746 613 605 -499 402 6 -119 73 438 -496 501 111 -273 380 205 -282 163 120 -226 -330 163 -240 -503 -217 -407 67 -180 552 54 -41 313 102 -81 -131 -336 362] [478 53 372 -764 -350 111 180 333 433 -253 406 144 110 221 106 -451 248 197 -660 659 549 369 52 424 -305 -183 108 -220 -645 -28 137 56 -77 -324 -161 489 378 -245 97 407 -137 -504 -342 570 -403 528 315 -411 -370 67 399 387 414 -341 -77 80 66 41 865 -407 207 -361 698 285 -41 -402 177 86 -729 -304 -730 31 -628 472 -591 48 -672 -200 -545 508 -435 173 -625 -43 -539 26 458 -81 -329 -141 -203 -120 -217 -79 41 375 -300 -202 329 236 -255 352 -543 314 -225 -562 249 -965 254 758 347 -217 105 714 57 -569 -208 786 220 -233 -124 782 -105 -60 284 -363 221 378 -263 -8] [83 -250 -126 186 -277 79 -24 367 320 228 153 -216 -332 -509 -75 -493 -2 -682 -196 50 30 11 542 837 -72 72 -680 847 65 -118 373 -455 312 -229 764 450 -580 -949 -300 -143 551 -269 856 443 -337 -101 263 -818 -192 -26 -825 -359 -523 74 535 -262 332 -465 127 -130 -108 272 -576 -75 89 -47 -133 1 -41 -178 144 -113 178 -459 335 -579 -324 816 119 644 -114 -785 644 -56 -720 -351 -643 -217 344 582 -308 194 1096 244 -242 69 -412 -394 1014 409 -126 -526 306 -263 366 -374 -112 -37 -505 -597 -617 75 -210 -47 -430 -332 183 -96 289 20 307 -32 -220 454 186 432 -276 331 -82 938] [-345 -127 645 83 -494 -393 447 -315 77 -119 277 686 -717 -355 139 119 -346 -436 84 -31 113 -610 -665 410 -771 308 -420 14 -261 187 -358 -386 -136 -621 425 -167 -507 134 -296 26 606 -499 -372 -45 -55 -335 70 -46 -149 -52 -371 -144 -647 -218 -228 42 216 -91 -436 -577 -619 170 499 -305 649 -318 -246 626 -16 392 399 797 -439 262 561 -239 41 -355 28 -106 -360 -104 -956 -357 276 -429 29 701 754 638 -356 -253 -166 -681 36 -24 29 418 881 601 -864 345 -478 571 472 -95 872 360 367 -434 51 170 509 -157 159 99 -515 306 -1295 -43 -649 -56 169 -141 -285 -159 -935 -404 -559 561] [-172 -238 121 -423 189 78 -279 -266 231 249 -215 446 283 -771 348 150 -149 -374 48 -95 6 954 1028 198 248 47 51 -445 -90 -564 268 -145 209 49 177 233 373 -578 -34 1045 -47 898 496 217 -703 -235 -418 -720 -346 147 296 93 -28 -93 -69 -238 138 28 357 -675 -152 181 -9 -501 -324 269 -718 356 126 -482 865 308 775 -511 -353 374 354 123 77 -1024 -498 -268 177 -190 146 -184 815 -257 165 261 -288 545 -70 340 -613 161 36 476 213 233 -160 -234 167 -47 120 -24 356 -572 -552 -597 621 202 -229 348 -40 290 610 233 372 -295 245 -122 382 253 215 53 -776 551 282 -206] [-943 -13 -708 421 571 -837 -1 -477 134 -183 -283 95 -29 697 -57 1142 282 83 336 -814 365 16 197 -565 513 -457 -43 -325 -484 -817 -344 279 188 177 -380 -138 597 645 379 -179 -504 307 -558 -148 368 -491 72 -294 211 313 324 447 271 -311 128 32 -51 489 674 -241 -949 178 231 -117 86 77 215 15 -489 557 -12 -167 938 398 -293 101 561 -198 19 -285 364 601 -52 1085 161 581 93 -502 26 -129 -359 121 -58 -355 154 -426 283 -125 -751 -474 374 93 -64 706 -187 -124 354 -110 -351 598 52 -139 -47 -518 13 378 -15 402 101 -57 -337 -361 -133 -260 -302 -139 -469 -663 558 -308] [491 217 -420 617 -488 263 -345 -379 151 -205 -787 707 1189 640 499 -160 252 -98 -127 -664 -28 -436 265 395 28 351 11 -395 -60 149 1 -190 -925 521 -476 -752 -10 398 36 -105 433 -36 226 202 -395 -511 496 242 -91 154 -206 437 230 -172 -293 -371 1273 -266 -539 -770 346 -480 -37 -51 678 688 12 -666 38 25 -432 -65 309 -117 333 118 -376 -769 -818 -1049 -215 582 142 135 -342 -474 -345 219 388 148 169 246 -546 -324 -17 50 581 -328 821 129 502 108 150 40 244 1067 -537 744 -19 177 592 264 -39 -266 92 248 -1247 -337 -129 -49 -632 -46 -17 346 -350 -605 -214 -231 -207 90] [231 298 251 -363 -382 -352 -371 293 2 649 -448 497 494 -535 404 54 -55 428 -301 99 -653 -2 323 747 91 318 -55 -430 417 198 451 488 -1079 -312 433 -293 499 -66 150 613 210 -558 -127 -626 -221 -131 155 454 -611 -132 -180 214 339 797 228 71 380 292 -478 335 69 233 377 -499 -68 111 677 253 912 101 784 -32 -12 160 -398 233 441 -518 -504 509 -56 -378 -809 -662 316 -349 -46 925 -383 -72 274 -274 -574 359 273 -431 -144 4 -695 1178 672 439 11 392 -501 753 343 -432 -648 410 361 597 696 -473 918 530 -444 632 111 483 -171 -22 184 -219 -202 232 -14 452 -622 27] [-717 595 253 -15 -47 -225 -130 -60 109 174 655 -237 -51 -270 311 54 -229 -741 -32 -10 -233 -8 -405 419 48 11 79 349 -621 377 -671 108 -102 316 -519 -144 -274 379 434 128 -738 128 -568 -700 605 -242 -377 806 154 -412 -78 -13 69 -514 -290 -253 -552 7 362 -417 156 312 137 61 -420 368 -17 -796 -644 -462 -110 352 -166 12 213 -465 -118 305 162 9 -313 355 -11 385 -10 589 581 16 445 -585 -20 -382 561 -108 540 421 -91 47 -79 55 267 -2 155 -229 464 28 -276 -27 -198 1 -353 -282 -415 843 -105 -284 -223 573 362 -136 -87 -571 249 -84 -69 -801 -595 -285 30 212] [119 -520 5 -157 -102 883 -94 -244 -299 738 45 -190 134 -218 206 149 166 -426 -38 7 66 209 104 599 -185 400 -438 -663 36 -42 -221 -797 581 369 -158 -351 107 396 239 28 -66 98 -215 -595 -397 -707 109 6 -37 54 11 1154 -156 72 -255 -795 706 -618 288 -80 125 -278 566 349 -185 1032 -494 -282 -399 -272 -441 -430 491 26 1044 -450 239 -12 296 -44 364 166 -177 43 44 382 -628 80 343 -630 503 -193 377 -19 -3 135 20 133 140 -188 121 -511 149 204 -824 183 157 79 520 277 289 -627 36 60 -366 -66 -139 431 36 150 -27 485 5 354 -756 351 159 -179 88 -183] [525 33 -239 274 -493 -388 -804 -132 -534 -356 611 -70 -290 -591 -49 645 -487 -33 -511 268 606 -809 -127 507 282 -62 -465 253 -338 -212 -166 718 30 -335 732 -136 142 -734 57 295 -375 1224 69 101 -143 55 339 34 492 236 299 -284 384 219 175 -797 -204 249 3 -761 827 -62 -691 -877 625 -574 -393 -415 -467 213 -82 -260 -463 -130 339 346 -723 -469 -818 537 -447 614 637 456 -525 267 -635 -474 -23 104 -401 -88 347 289 470 1019 306 -662 269 -261 -711 128 -341 -149 -242 522 -684 137 -646 -70 53 -2 -570 138 -201 -490 -110 -120 302 69 47 171 599 359 205 -938 198 -443 184 -300] [357 70 998 160 299 -296 602 194 -370 156 328 -268 -81 575 -128 108 -564 -317 200 -414 -196 -754 -344 201 -251 131 -75 218 201 -790 -507 -162 -980 687 -108 -941 -87 -652 141 -548 552 -725 17 -367 -208 -156 339 -847 -220 -18 -186 -141 -636 -328 302 -226 85 -28 -836 -686 -880 -74 249 -170 -908 -288 530 84 -307 -452 358 390 -363 -471 113 -412 453 -60 -381 -252 -314 -328 845 -93 122 -768 496 -267 -194 -210 20 -812 617 -678 -349 346 -387 39 529 134 -44 -146 166 108 176 85 478 -492 -324 166 10 707 35 -404 147 -405 -335 -452 -381 447 276 -161 227 78 -36 -321 -779 660 -140 530] [-361 10 -461 -361 -309 291 562 418 -482 -482 -131 -142 -833 554 -116 -169 -810 641 -241 990 -324 840 -429 446 500 438 -207 122 83 188 -177 -355 1159 48 158 391 -1035 -284 91 -389 271 -104 -239 -58 -66 -97 801 244 1205 89 -87 151 -286 -144 287 46 -530 397 13 -160 426 201 -527 191 949 -16 -556 -344 298 -253 5 -199 89 36 677 622 347 251 212 470 -736 103 -242 551 -297 260 -470 84 -184 -342 -999 397 -685 546 543 -74 133 526 -432 -61 214 -132 261 -541 391 157 362 343 -482 -264 461 195 621 240 162 -340 215 -608 22 208 1035 309 -294 -233 24 310 307 -73 614 -437] [-13 183 -568 557 -162 35 -149 -282 -675 -42 -203 116 34 -243 -103 -128 248 279 -563 -167 -424 216 627 -128 190 115 429 -481 -317 743 545 526 120 202 46 309 -38 326 103 -422 685 456 -87 396 -110 36 -136 -29 314 -37 -100 -57 -141 157 -549 560 187 33 258 -203 834 276 -117 -178 -65 -43 -893 33 -33 -437 673 520 426 183 458 368 -120 -154 321 -375 -447 -82 624 -389 -452 -440 -588 687 342 526 -224 873 -328 410 98 -111 -375 -666 -114 -501 -325 -642 -229 -722 -439 301 19 494 101 -90 -38 289 57 257 -914 -316 -460 -134 225 -1170 340 27 -22 -15 -569 8 1125 175 -339 -288] [-216 1152 -77 69 58 116 -517 143 845 35 -438 -30 465 260 676 -364 572 105 -125 178 353 -16 -186 40 -40 66 41 38 118 587 61 -429 -572 -545 -271 -65 250 -866 -596 -164 602 -463 -247 -380 154 709 22 -578 -1001 211 -481 -188 784 295 -536 558 768 -143 190 572 276 -634 754 783 452 40 317 -316 109 396 -506 -157 -133 322 -340 370 -244 -433 234 -347 176 -495 -664 -478 -94 -12 427 -79 250 -168 345 -447 -148 -122 450 -287 226 488 557 600 -246 125 559 -874 302 -130 -195 7 632 36 -633 377 690 130 404 -136 -143 -413 -411 82 308 698 -5 -31 -234 302 255 542 -733 425] [356 301 1052 -100 -67 -631 619 -414 -253 688 385 424 -194 -616 262 266 44 67 317 313 -323 287 391 328 -140 8 -283 -655 -181 -88 565 630 -899 371 -414 27 447 93 131 226 -54 100 -315 -373 -397 31 216 370 -121 176 -100 -270 358 -55 -542 271 -311 583 -788 -82 83 -30 719 -181 296 -495 90 548 781 -629 1326 -293 -386 -22 -343 -403 126 18 -290 -331 -182 209 -420 -738 331 -378 -339 42 -146 -221 34 -236 -323 155 -324 246 -65 347 -425 370 -154 189 -769 364 63 -734 -480 -860 -133 -126 97 -40 -210 41 190 -129 10 809 -206 464 370 359 840 -46 1050 -497 -715 69 2 -132] [332 123 -403 459 -467 -70 526 -288 -245 -710 874 -179 23 -372 -271 -226 226 226 239 -661 57 -701 -520 -481 -431 -824 -210 -375 70 -403 19 -720 606 1132 -800 -317 417 -677 303 68 38 178 -569 186 24 354 -268 -204 300 -362 458 46 -716 -39 396 -345 300 119 -458 -21 -227 -606 222 -304 470 571 205 -161 -845 -256 -633 -514 436 -146 500 -407 265 -265 1049 -488 241 -46 538 55 247 260 -227 -250 568 -36 -75 96 35 -912 -80 299 1114 -105 228 -159 -33 70 -747 517 -146 424 -572 344 613 312 78 -68 -97 -837 -130 -8 72 -50 -133 309 -271 -157 286 297 -636 -314 73 -783 588 20] [-82 135 312 26 215 -881 250 134 -510 534 236 -480 417 73 669 900 486 -206 268 -739 904 -793 -564 -579 -611 82 366 -467 548 -97 -433 -126 -394 217 -96 -786 54 -506 -257 -765 615 113 -55 93 1 -879 -495 171 -240 -386 1012 -165 25 -35 -248 508 583 -366 -161 686 -165 -475 387 109 266 -87 9 108 -406 455 -688 -32 198 326 7 395 3 -53 -278 250 214 1027 302 -213 422 -128 -13 24 154 -367 287 -52 -636 19 -302 585 341 239 -75 -98 -845 -333 364 -48 -460 172 -485 -79 855 -195 -882 -390 -271 93 144 -201 142 -359 -961 476 -420 -200 265 368 -22 -643 -378 -674 388 -760] [-19 -121 -96 82 634 -203 478 436 352 -318 686 -571 -384 210 195 305 -51 -169 318 -508 206 -23 -773 -493 388 12 222 220 544 -306 708 1068 -70 -92 -631 369 -276 377 -702 -210 -71 -671 -497 -59 243 -76 -46 509 -118 -378 890 245 433 -352 -306 204 -409 -32 -183 309 228 562 209 90 130 155 361 -143 -16 439 -655 -126 279 108 517 -426 -922 -377 288 22 -404 572 789 -247 -417 -228 -18 9 -195 -257 248 -245 18 170 -135 -226 74 -715 -200 808 -452 263 525 -612 633 -67 -306 129 99 515 21 8 197 -166 -388 -225 218 -926 -170 -320 -556 460 56 -48 80 -138 715 -287 -250 490]

Bibliography

[AD21] Martin Albrecht and Léo Ducas. "Lattice Attacks on NTRU and LWE: A History of Refinements". In: *Cryptology ePrint Archive: Report 2021/799* (2021).

[Ajt96] Miklós Ajtai. "Generating hard instances of lattice problems". In: *Symposium on Theory of Computing (STOC 1996)*. ACM. 1996, pp. 99–108.

[AKS01] Miklós Ajtai, Ravi Kumar, and Dandapani Sivakumar. "A sieve algorithm for the shortest lattice vector problem". In: *Symposium on Theory of Computing (STOC 2001)*. ACM. 2001, pp. 601–610.

[Alb+18] Martin R Albrecht et al. "Estimate all the {LWE, NTRU} schemes!" In: *Security and Cryptography for Networks (SCN 2018)*. Vol. 11035. Lecture Notes in Computer Science. 2018, pp. 351–367.

[Alb+19] Martin Albrecht et al. "The general sieve kernel and new records in lattice reduction". In: *Advances in Cryptology–EUROCRYPT 2019*. Vol. 11477. Lecture Notes in Computer Science. Springer. 2019, pp. 717–746.

[Bab86] László Babai. "On Lovász' lattice reduction and the nearest lattice point problem". In: *Combinatorica* 6.1 (1986), pp. 1–13.

[BBK19] Michael Burger, Christian Bischof, and Juliane Krämer. "p3Enum: A New Parameterizable and Shared-Memory Parallelized Shortest Vector Problem Solver". In: *Computational Science–ICCS 2019*. Vol. 11540. Lecture Notes in Computer Science. Springer. 2019, pp. 535–542.

[BG73] Åke Björck and Gene H Golub. "Numerical methods for computing angles between linear subspaces". In: *Mathematics of computation* 27.123 (1973), pp. 579–594.

[BN02] Alexander Barg and D Yu Nogin. "Bounds on packings of spheres in the Grassmann manifold". In: *IEEE Transactions on Information Theory* 48.9 (2002), pp. 2450–2454.

[Bre11] Murray R Bremner. *Lattice basis reduction: An introduction to the LLL algorithm and its applications*. CRC Press, 2011.

[BSW18] Shi Bai, Damien Stehlé, and Weiqiang Wen. "Measuring, Simulating and Exploiting the Head Concavity Phenomenon in BKZ". In: *Advances in Cryptology – ASIACRYPT 2018*. Vol. 11272. Lecture Notes in Computer Science. Springer, 2018, pp. 369–404. DOI: 10.1007/978-3-030-03326-2_13.

[Cai00] Jin-Yi Cai. "The Complexity of Some Lattice Problems". In: *Algorithmic Number Theory*. Ed. by Wieb Bosma. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 1–32. ISBN: 978-3-540-44994-2.

[Che13] Yuanmi Chen. "Réduction de réseau et sécurité concrete du chiffrement completement homomorphe". PhD thesis. Paris 7, 2013.

[Che16] Hao Chen. "A Measure Version of Gaussian Heuristic". In: *IACR Cryptology ePrint Archive: Report 2016/439* (2016).

[CN11] Yuanmi Chen and Phong Q Nguyen. "BKZ 2.0: Better lattice security estimates". In: *Advances in Cryptology–ASIACRYPT 2011*. Vol. 7073. Lecture Notes in Computer Science. Springer. 2011, pp. 1–20.

[Det+10] Jérémie Detrey et al. "Accelerating lattice reduction with FPGAs". In: *International Conference on Cryptology and Information Security in Latin America*. Springer. 2010, pp. 124–143.

[DG96] Peter Deutsch and Jean-Loup Gailly. *Zlib compressed data format specification version 3.3*. Tech. rep. RFC 1950, May, 1996.

[DS10] Öz6Kür Dagdelen and Michael Schneider. "Parallel enumeration of shortest lattice vectors". In: *Euro-Par 2010–Parallel Processing*. Vol. 6272. Lecture Notes in Computer Science. Springer. 2010, pp. 211–222.

[DSW21] Léo Ducas, Marc Stevens, and Wessel van Woerden. "Advanced Lattice Sieving on GPUs, with Tensor Cores". In: *IACR ePrint 2021/141* (2021).

[Duc18] Léo Ducas. "Shortest vector from lattice sieving: A few dimensions for free". In: *Adavances in Cryptology–EUROCRYPT 2018*. Vol. 10820. Lecture Notes in Computer Science. Springer. 2018, pp. 125–145.

[EAS98] Alan Edelman, Tomás A Arias, and Steven T Smith. "The geometry of algorithms with orthogonality constraints". In: *SIAM journal on Matrix Analysis and Applications* 20.2 (1998), pp. 303–353.

[Fuj+21] Koichi Fujii et al. *Solving Challenging Large Scale QAPs*. eng. Tech. rep. 21-02. Takustr. 7, 14195 Berlin: ZIB, 2021.

[Gam+17] Gerald Gamrath et al. "SCIP-Jack—a solver for STP and variants with parallelization extensions". In: *Mathematical Programming Computation* 9.2 (2017), pp. 231–296. DOI: 10.1007/s12532-016-0114-x.

[GM03] Daniel Goldstein and Andrew Mayer. "On the equidistribution of Hecke points". In: *Forum Mathematicum*. Vol. 15. 2. De Gruyter. 2003, pp. 165–190.

[GN08] Nicolas Gama and Phong Q Nguyen. "Predicting lattice reduction". In: *Advances in Cryptology–EUROCRYPT 2008*. Vol. 4965. Lecture Notes in Computer Science. Springer. 2008, pp. 31–51.

[GNR10] Nicolas Gama, Phong Q. Nguyen, and Oded Regev. "Lattice Enumeration Using Extreme Pruning". In: *Advances in Cryptology – EUROCRYPT 2010*. Ed. by Henri Gilbert. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 257–278. ISBN: 978-3-642-13190-5.

[GVL96] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Forth. The Johns Hopkins University Press, 1996.

[Her+10] Jens Hermans et al. "Parallel shortest lattice vector enumeration on graphics cards". In: *Progress in Cryptology–AFRICACRYPT 2010*. Vol. 6055. Lecture Notes in Computer Science. Springer. 2010, pp. 52–68.

[Her50] C. Hermite. "Extraits de lettres de M. Hermite à M. Jacobi sur différents objets de la théorie des nombres: Deuxième lettre". In: *Journal für die Reine und Angewandte Mathematik* (1850), pp. 279–315.

[Jou12] Antoine Joux. "A tutorial on high performance computing applied to cryptanalysis (invited talk)". In: *Advances in Cryptology–EUROCRYPT 2012*. Vol. 7237. Lecture Notes in Computer Science. Springer. 2012, pp. 1–7.

[Kan87] Ravi Kannan. "Minkowski's convex body theorem and integer programming". In: *Mathematics of operations research* 12.3 (1987), pp. 415–440.

[Kle00] Philip Klein. "Finding the closest lattice vector when it's unusually close". In: *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*. 2000, pp. 937–941.

[Kuo+11] Po-Chun Kuo et al. "Extreme Enumeration on GPU and in Clouds". In: *Cryptographic Hardware and Embedded Systems–CHES 2011*. Vol. 6917. Lecture Notes in Computer Science. Springer. 2011, pp. 176–191.

[LLL82] Arjen Klaas Lenstra, Hendrik Willem Lenstra, and László Lovász. "Factoring polynomials with rational coefficients". In: *Mathematische Annalen* 261.4 (1982), pp. 515–534.

[LLS90] J. C. Lagarias, H. W. Lenstra, and C. P. Schnorr. "Korkin-Zolotarev bases and successive minima of a lattice and its reciprocal lattice". In: *Combinatorica* 10.4 (Dec. 1990), pp. 333–348. DOI: 10.1007/BF02128669. URL: https://doi.org/10.1007/BF02128669.

[Mic01] Daniele Micciancio. "The shortest vector in a lattice is hard to approximate to within some constant". In: *SIAM journal on Computing* 30.6 (2001), pp. 2008–2035. DOI: 10.1137/S0097539700373039.

[Mun+19] Lluís-Miquel Munguía et al. "Parallel PIPS-SBB: multi-level parallelism for stochastic mixed-integer programs". In: *Computational Optimization and Applications* 73.2 (June 2019), pp. 575–601. ISSN: 1573-2894. DOI: 10.1007/s10589-019-00074-0. URL: https://doi.org/10.1007/s10589-019-00074-0.

[MV10] Daniele Micciancio and Panagiotis Voulgaris. "Faster exponential time algorithms for the shortest vector problem". In: *Symposium on Discrete Algorithms (SODA 2010)*. ACM-SIAM. 2010, pp. 1468–1480.

[Ngu09] Phong Q Nguyen. "Hermite's constant and lattice algorithms". In: *The LLL Algorithm*. Springer, 2009, pp. 19–69.

[Pei16] Chris Peikert. "A Decade of Lattice Cryptography". In: *Foundations and Trends in Theoretical Computer Science* 10.4 (2016), pp. 283–424. ISSN: 1551-305X. DOI: 10.1561/0400000074. URL: http://dx.doi.org/10.1561/0400000074.

[Poh87] Michael Pohst. "A modification of the LLL reduction algorithm". In: *Journal of Symbolic Computation* 4.1 (1987), pp. 123–127.

[PSZ21] Simon Pohmann, Marc Stevens, and Jens Zumbrägel. "Lattice Enumeration on GPUs for fplll". In: *IACR ePrint 2021/430* (2021).

[Ral+18] Ted Ralphs et al. "Parallel Solvers for Mixed Integer Linear Optimization". In: *Handbook of Parallel Constraint Reasoning*. Ed. by Youssef Hamadi and Lakhdar Sais. Cham: Springer International Publishing, 2018, pp. 283–336. ISBN: 978-3-319-63516-3. DOI: 10.1007/978-3-319-63516-3{_}8. URL: https://doi.org/10.1007/978-3-319-63516-3_8.

[RSK21] Daniel Rehfeldt, Yuji Shinano, and Thorsten Koch. "SCIP-Jack: An Exact High Performance Solver for Steiner Tree Problems in Graphs and Related Problems". In: *Modeling, Simulation and Optimization of Complex Processes HPSC 2018*. Ed. by Hans Georg Bock et al. Cham: Springer International Publishing, 2021, pp. 201–223. ISBN: 978-3-030-55240-4.

[SBH18] Yuji Shinano, Timo Berthold, and Stefan Heinz. "ParaXpress: an experimental extension of the FICO Xpress-Optimizer to solve hard MIPs on supercomputers". In: *Optimization Methods and Software* 33.3 (2018), pp. 530–539. DOI: 10.1080/10556788.2018.1428602. eprint: https://doi.org/10.1080/10556788.2018.1428602. URL: https://doi.org/10.1080/10556788.2018.1428602.

[Sch03] Claus Peter Schnorr. "Lattice reduction by random sampling and birthday methods". In: *Symposium on Theoretical Aspects of Computer Science (STACS 2003)*. Vol. 2607. Lecture Notes in Computer Science. Springer. 2003, pp. 145–156.

[Sch+10] Michael Schneider et al. "SVP challenge (2010)". In: *URL: http://latticechallenge.org/svp-challenge* (2010).

[Sch87] Claus-Peter Schnorr. "A hierarchy of polynomial time lattice basis reduction algorithms". In: *Theoretical computer science* 53.2-3 (1987), pp. 201–224.

[Sch92] Claus-Peter Schnorr. *Block Korkin-Zolotarev bases and successive minima*. International Computer Science Institute, 1992.

[Sci] *SCIP: Solving Constraint Integer Programs*. http://scip.zib.de/.

[SE94] Claus-Peter Schnorr and Martin Euchner. "Lattice basis reduction: Improved practical algorithms and solving subset sum problems". In: *Mathematical programming* 66 (1994), pp. 181–199.

[Shi+11] Yuji Shinano et al. "ParaSCIP—a parallel extension of SCIP". In: *Competence in High Performance Computing 2010*. Springer, 2011, pp. 135–148.

[Shi+16] Yuji Shinano et al. "Solving Open MIP Instances with ParaSCIP on Supercomputers Using up to 80,000 Cores". In: *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. Los Alamitos, CA, USA: IEEE Computer Society, 2016, pp. 770–779.

[Shi+18a] Yuji Shinano et al. "FiberSCIP—a shared memory parallelization of SCIP". In: *INFORMS Journal on Computing* 30.1 (2018), pp. 11–30.

[Shi+18b] Yuji Shinano et al. "FiberSCIP—A Shared Memory Parallelization of SCIP". In: *INFORMS Journal on Computing* 30.1 (2018), pp. 11–30. DOI: 10.1287/ijoc.2017.0762. eprint: https://doi.org/10.1287/ijoc.2017.0762. URL: https://doi.org/10.1287/ijoc.2017.0762.

[Sho94] Peter W. Shor. "Algorithms for quantum computation: Discrete logarithms and factoring". In: *Symposium on Foundations of Computer Science (FOCS 1994)*. IEEE, 1994, pp. 124–134.

[SN] The National Institute of Standards and Technology (NIST). "Post-Quantum Cryptography". URL: https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization.

[SRG19] Yuji Shinano, Daniel Rehfeldt, and Tristan Gally. "An Easy Way to Build Parallel State-of-the-art Combinatorial Optimization Problem Solvers: A Computational Study on Solving Steiner Tree Problems and Mixed Integer Semidefinite Programs by using ug[SCIP-*,*]-Libraries". In: *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 2019, pp. 530–541. DOI: 10.1109/IPDPSW.2019.00095.

[SRK19] Yuji Shinano, Daniel Rehfeldt, and Thorsten Koch. "Building Optimal Steiner Trees on Supercomputers by Using up to 43,000 Cores". In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research. CPAIOR 2019*. Vol. 11494. 2019, pp. 529–539. DOI: 10.1007/978-3-030-19212-9_35.

[Tat+20] Nariaki Tateiwa et al. "Massive parallelization for finding shortest lattice vectors based on ubiquity generator framework". In: *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE. 2020, pp. 1–15.

[Tat+21] Nariaki Tateiwa et al. "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)*. IEEE. 2021.

[The16] The FPLLL development team. "fplll, a lattice reduction library". 2016. URL: https://github.com/fplll/fplll.

[TKH18] Tadanori Teruya, Kenji Kashiwabara, and Goichiro Hanaoka. "Fast lattice basis reduction suitable for massive parallelization and its application to the shortest vector problem". In: *Public Key Cryptography (PKC 2018)*. Vol. 10769. Lecture Notes in Computer Science. Springer. 2018, pp. 437–460.

[Ug] *UG: Ubiquity Generator framework*. http://ug.zib.de/.

[Yas21] Masaya Yasuda. "A Survey of Solving SVP Algorithms and Recent Strategies for Solving the SVP Challenge". In: *International Symposium on Mathematics, Quantum Theory, and Cryptography*. Springer. 2021, pp. 189–207.

[YD17] Yang Yu and Léo Ducas. "Second order statistical behavior of LLL and BKZ". In: *Selected Areas in Cryptography (SAC 2017)*. Vol. 10719. Lecture Notes in Computer Science. Springer. 2017, pp. 3–22.

[YNY20] Masaya Yasuda, Satoshi Nakamura, and Junpei Yamaguchi. "Analysis of Deep-BKZ reduction for finding short lattice vectors". In: *Designs, Codes and Cryptography* 88 (2020), pp. 2077–2100.

[YY17] Junpei Yamaguchi and Masaya Yasuda. "Explicit formula for Gram-Schmidt vectors in LLL with deep insertions and its applications". In: *Number-Theoretic Methods in Cryptology (NuTMiC 2017)*. Vol. 10737. Lecture Notes in Computer Science. Springer. 2017, pp. 142–160.

[YY19] Masaya Yasuda and Junpei Yamaguchi. "A new polynomial-time variant of LLL with deep insertions for decreasing the squared-sum of Gram-Schmidt lengths". In: *Designs, Codes and Cryptography* 87 (11 2019), pp. 2489–2505.

List of Figures

List of Tables