Vol.2020-HPC-176 No.1
2020/9/25

最短格子ベクトル問題求解における Ubiquity Generator Framework を用いた大規模 MPI 並列化

立岩 斉明[1,a)]　品野 勇治[2,b)]　吉田 明広[1,c)]　鍛冶 静雄[1,d)]　安田 雅哉[3,e)]　藤澤 克樹[1,f)]

概要：格子暗号は，古典計算機や量子計算機からの攻撃に対しても安全であると考えられており，次世代の暗号化技術として注目されている. その本質的な安全性は，最短ベクトル問題 (SVP) の求解困難性に依存しており，セキュリティレベルを決定するために大規模計算機により SVP の難易度を正確に推定することが重要である. 本研究では，世界初の分散型非同期並列 SVP ソルバーである MAssively Parallel solver for SVP (MAP-SVP) を開発した. MAP-SVP は分枝限定ソルバの汎用ソフトフェアである Ubiquity Generator framework を適用することで非同期的に情報を共有しながら SVP アルゴリズムを大規模に並列実行することが可能になった. MAP-SVP の性能を実証するために Darmstadt SVP チャレンジのインスタンスを複数解き，最大 91,200 のコア数を用いた実行により 127 次元の記録を更新した.

キーワード：最短ベクトル問題，並列計算，Ubiquity Generator framework，格子基底簡約，列挙法，格子暗号

Massive MPI Parallelization for Solving Shortest Vector Problem Based on Ubiquity Generator Framework

Abstract: Lattice-based cryptography is considered the next generation encryption technique because it is believed to be secure against attacks from classical and quantum computers. Its security depends essentially on the difficulty of solving the shortest vector problem (SVP). It is important to estimate the hardness of solving SVP by high-performance computers to determine the security level. In this study, we develop the world's first distributed and asynchronous parallel SVP solver, the MAssively Parallel solver for SVP (MAP-SVP). This solver can parallelize the SVP algorithm on a large scale while sharing information asynchronously by applying the Ubiquity Generator framework, which is a generic framework for branch-and-bound algorithms. We demonstrate the performance of the MAP-SVP by solving instances of the Darmstadt SVP challenge, and MAP-SVP marked a new record at 127 dimension by using up to 91, 200 cores.

Keywords: The shortest vector problem, Parallel computation, Ubiquity Generator framework, Lattice basis reduction, ENUM, Lattice-based cryptography

1. はじめに

自然数 n に対し，Euclid 空間 $\mathbb{R}^n$ における一次独立な n 個のベクトル $\mathbf{b}_1, \ldots, \mathbf{b}_n$ の整数係数の線形結合全体 L を n 次元の格子という. 近年，暗号分野において格子は非常注目されている. 現在普及している RSA 暗号や楕円曲線暗号は Shor のアルゴリズム [1] により大規模な量子計算機が実現すれば解読できてしまうため，量子計算機による解読にも耐性を持つ情報セキュリティシステムの開発は喫緊の課題である. この流れを受けて，米国標準技術研究所 NIST は量子計算機による解読に耐性を持つポスト量子暗号（または耐量子計算機暗号）の標準化計画を 2015 年に開始した [2]. 2020 年 7 月中旬には，量子暗号標準計画は第 3 ラウンドまで進み，受諾する暗号システムは 7 件に絞りこまれたが，その内の 5 件が格子に基づく暗号システム (格子

[1] 九州大学
[2] Zuse Institute Berlin (ZIB)
[3] 立教大学
[a)] n-tateiwa@kyudai.jp
[b)] shinano@zib.de
[c)] akihiro.yoshida.916@kyudai.jp
[d)] skaji@imi.kyushu-u.ac.jp
[e)] myasuda@rikkyo.ac.jp
[f)] fujisawa@imi.kyushu-u.ac.jp

暗号システム) であった.

格子における計算問題として最短ベクトル問題（Shortest Vector Problem, SVP）が最も有名で，それは格子の基底が与えられたとき，最も小さいノルムを持つ非零格子ベクトルを求めるというものである. 暗号分野においては，最短ベクトル問題の計算困難性は格子暗号の安全性を支えている. 最短ベクトル問題の求解アルゴリズムの計算時間や性能を評価する目的で，2010 年からドイツ・Darmstadt 工科大が主催する SVP チャレンジ [3] において 40 から 200 次元の格子基底が web 公開され，世界中の研究者が SVP 解読の記録更新を競い合っている. 具体的には，各次元の格子 L において Euclid ノルムが $1.05 \cdot \mathrm{GH}(L)$ より小さい格子ベクトルを見つけた場合，SVP チャレンジに投稿することができる.（ただし，$\mathrm{GH}(L)$ は格子 L における最短な非零ベクトルのノルムの期待値である.）つまり，SVP チャレンジは最短ベクトル問題の厳密解ではなく近似解を見つけるコンテストである.

格子の最短な非零ベクトルを見つけるアルゴリズムとして篩法 (Sieve) と列挙法（ENUM）がある. これらのアルゴリズムは共に短い格子ベクトルを全数探索するが，その探索数は格子次元に対して指数関数的に増大する. 特に，列挙法は確定的アルゴリズムであるのに対し，篩法は確率的アルゴリズムで高次元の格子において列挙法より高速である. 実際，SVP チャレンジにおいて 130 次元以上の高次元の記録の多くは篩法の変種で見つけられたものである. 篩法では 2 つの短い格子ベクトルの差分からより短い格子ベクトルを見つけていくため，高次元の最短ベクトル問題では膨大な数のベクトルを格納し，格子次元に関して指数関数的なメモリが必要となる. 実際，篩法に基づく最新の SVP 解読システム G6K （General Sieve Kernel [4]）では 127 次元の最短ベクトル問題に対し約 246 GBytes のメモリが必要と報告されている. 並列数を増やしたとしても，メモリ要件の観点からより高次元の最短ベクトル問題を解く際に篩法を利用することは困難である. 一方，列挙法は篩法より高い漸近計算量を持つが，格子次元に関して多項式的な空間計算量を持ち，より高次元の最短ベクトル問題を解くのに有用である.

本研究では，大規模並列計算が可能な新しい SVP ソルバである ***M**assively **P**arallel Solver for **SVP*** (*MAP-SVP*) [5] を提案する. MAP-SVP は世界初の実用的な分散非同期 MPI 並列ソルバであり，最大 127 次元の SVP チャレンジの記録更新に成功した. これは厳密解法ソルバで達成された最短ベクトル記録の中では，最大次元の記録である (2020 年 7 月現在). また記録更新にあたり最大 91,200 コア数 (=MPI プロセス数) での実行を行い，我々の知る限りでは SVP 解読実験における最も大規模な並列環境での実行である. MAP-SVP は複数のプロセスが，並列に 2 種類の SVP 解法，ENUM と DeepBKZ を実行する. 各プロセスの空間計算量

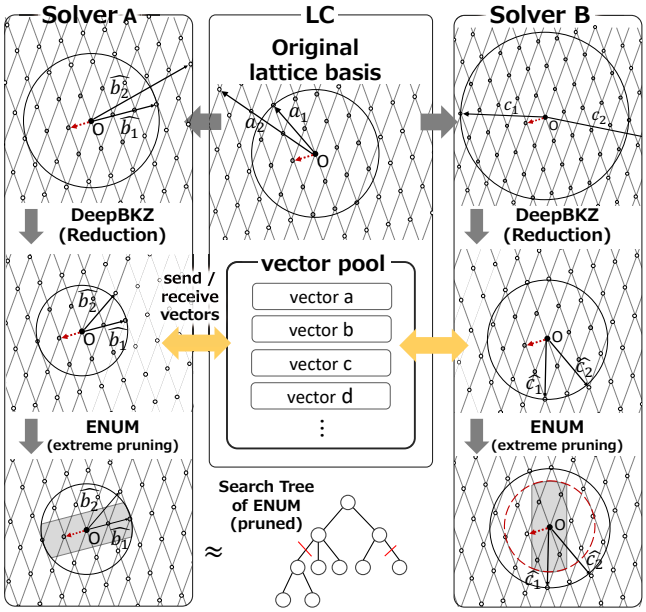

図 1　MAP-SVP の全体図
Fig. 1　Overview of our new SVP solver

は n 次元 SVP に対して $O(n^2)$ であり，実際我々の数値実験では 155 次元の SVP インスタンスでも 1 プロセスあたりのメモリ使用量は 0.013GB 以下であるため，より高次元の SVP に対してもメモリ要件は問題にならない. より詳細な性能評価は [5] に記載されている.

MAP-SVP の並列化機能は Ubiquity Generator (*UG*) Framework と UG に新たに作成した仮想関数である *parallelDispatch* をベースに実装されている. UG は分枝限定法ソルバを外側から並列化するための汎用ソフトフェアであり，8 万コアでの大規模 MPI 並列計算の動作実績を持つ [6]. UG の parallelDispatch によって，少ない通信オーバーヘッドで大量のプロセスが非同期に情報共有しながら安定的に動作させることが可能になった.

2 次元の SVP 解読における MAP-SVP の動作概要を図 1 に示す. 本システムは *LoadCoordinator* (*LC*) とよばれる管理プロセスと，複数の `Solver` で構成されている. 格子基底 **B** を SVP インスタンスとして LC に与えると，LC はそれを `Solver` に分配する. `Solver` は格子基底を受け取ると，それをランダムに変換した格子基底を生成する. 図中，実線の矢印，点，円はそれぞれ格子基底ベクトル，格子，ENUM の探索空間を表しており，円の半径は暫定最短ベクトルのノルムであり実行が進むにつれて小さくなる. 各 `Solver` は DeepBKZ と ENUM を実行し，その最中 LC が管理する *vector pool* を介して短い格子ベクトルを共有する. ENUM の前処理として DeepBKZ により格子基底を簡約することで ENUM の探索空間が削減される. また各 `Solver` の探索空間の半径は，図中 Solver B の赤丸の破線で示しているように，全 `Solver` の暫定最短ベクトルに合わせて随時縮小される. ENUM では *extreme pruning* [7] 手法を用い

て探索空間を確率的に削減する. 図中， 赤矢印が最短ベクトル， 灰色の領域が削減された探索空間を表している. 入力格子基底のランダマイズにより， 各 Solver の探索空間は一般的に異なっているため， 最短ベクトルが削減後の探索空間に含まれるものもあれば，排除されるものも存在する. MAP-SVP では， 高い確率で少なくとも 1 つの Solver の探索木に最短ベクトルが含まれるようにパラメーターを調節して Solver の探索木を刈り込むため， Solver 数の増加にほぼ反比例して 各 Solver の探索空間を小さくすることができる.

さらに，DeepBKZ と ENUM はチェックポイントとリスタート機能を備えるが，そのために Solver は最新の格子基底と ENUM 探索における最終ノード情報の保存および読み込みだけが必要であり， リスタートに必要な事前計算の時間は小さい.

本研究における貢献は以下のとおりである：通信オーバーヘッドが小さく大規模並列化に適しており， メモリ消費量が少なく， 優れたチェックポイントやリスタート機能を備えた，世界初の分散非同期 MPI 並列 SVP ソルバ MAP-SVP を開発したこと，開発のために一般的なフレームワークである UG を拡張してベクトルの非同期共有を可能にしたこと，大規模な数値実験を行い SVP チャレンジにて 104 次元、111 次元、121 次元、127 次元の新記録を達成したことである. 特に， 91,200 プロセスでの実験は SVP 研究では最大規模であり特筆に値する.

2. 数学的準備

本節では， 格子の基本的な性質を述べると共に，最短ベクトル問題を解くアルゴリズムについてまとめておく.

2.1 格子と基底

自然数 n に対して， Euclid 空間 $\mathbb{R}^n$ 内の n 個の一次独立な（列）ベクトルを $\mathbf{b}_1, \ldots, \mathbf{b}_n$ とする. この一次独立なベクトル整数係数による線形結合全体の集合

$$L = \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_n) := \left\{ \sum_{i=1}^{n} v_i \mathbf{b}_i : v_i \in \mathbb{Z}\ (1 \le \forall i \le n) \right\}$$

を n 次元の**格子**と呼び， $n \times n$-行列 $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ を格子 L の**基底**と呼ぶ. つまり，基底は正則行列であり，その列ベクトルが対応する格子を生成する. ある格子を生成する基底は一意ではなく， 2 次元以上の格子は無限に多くの異なる基底を持つ. 2 つの基底 $\mathbf{B}_1$ と $\mathbf{B}_2$ が同じ格子を張ることと， $\mathbf{B}_1 = \mathbf{B}_2\mathbf{U}$ を満たす $n \times n$-ユニモジュラ行列 $\mathbf{U}$ が存在することは同値である. ここで， **ユニモジュラ行列**とは， 行列式が ± 1 であるような整数成分を持つ正方行列のことである. 格子 L の**体積**は L の基底 $\mathbf{B}$ を用いて $\mathrm{vol}(L) = |\det(\mathbf{B})|$ と定める. 格子の体積は基底の取り方には依存しないことに注意する.

格子の基底 $\mathbf{B}$ に対する **Gram-Schmidt の直交化**ベクトル $\mathbf{B}^* = (\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*)$ を次のように帰納的に定義する：

$$\begin{cases} \mathbf{b}_1^* = \mathbf{b}_1, \\ \mathbf{b}_i^* = \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{i,j} \mathbf{b}_j^*, \quad \mu_{i,j} = \dfrac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\|\mathbf{b}_j^*\|^2} \quad (j < i). \end{cases}$$

ここで Gram-Schmidt の係数による $n \times n$-行列を $\mu = (\mu_{i,j})$ とする.（ただし，すべての $i < j$ に対し $\mu_{i,j} = 0$ とし，すべての対角成分を $\mu_{k,k} = 1$ とする.）このとき $\mathbf{B} = \mathbf{B}^*\mu$ が成り立ち，Gram-Schmidt ベクトルの直交性より $\mathrm{vol}(L) = \prod_{i=1}^{n} \|\mathbf{b}_i^*\|$ が成り立つ. 各 $1 \le \ell \le n$ に対して， Euclid 空間 $\mathbb{R}^n$ から $\mathbb{R}$-ベクトル空間 $\langle \mathbf{b}_1, \ldots, \mathbf{b}_{\ell-1} \rangle_{\mathbb{R}}$ の直交補空間への**直交射影**を

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

と定める. 直交射影は基底の取り方に依存することに注意する.

2.2 最短ベクトル問題

与えられた格子 L の基底 $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ から， 格子 L における最短な非零ベクトルを見つける問題を最短ベクトル問題 (SVP) という. (ランダム簡約の仮定下で)SVP は NP 困難であることが証明されている [8]. 暗号の観点から実用的には，近似因子 $\alpha \ge 1$ が与えられたとき， $\|\mathbf{z}\| \le \alpha \lambda_1(L)$ を満たす非零ベクトル $\mathbf{z} \in L$ を見つる近似問題が重要である. ここで $\lambda_1(L)$ は格子 L における最短な非零ベクトルのノルムを表し， **第一逐次最小**と呼ばれる.

2.2.1 第一逐次最小の期待値

Euclid 空間 $\mathbb{R}^n$ 内の n 次元格子 L と可測集合 S の共通部分 $L \cap S$ に含まれる格子ベクトルの個数はおよそ $\mathrm{vol}(S)/\mathrm{vol}(L)$ であると期待できる. これは **Gauss のヒューリスティック（Gaussian Heuristic）**と呼ばれる. 特に， 集合 S として格子 L の第一逐次最小 $\lambda_1(L)$ を半径に持つ n 次元開球をとることで， 逐次最小 $\lambda_1(L)$ の大きさはおおよそ

$$\mathrm{GH}(L) := \nu_n^{-\frac{1}{n}} \mathrm{vol}(L)^{\frac{1}{n}} \sim \sqrt{\frac{n}{2\pi e}} \mathrm{vol}(L)^{\frac{1}{n}} \qquad (1)$$

に一致すると期待できる.（ただし， ν_n は n 次元単位球の体積とする.）これは多くの場合に成り立つと信じられている経験的事実で， 高次元のランダムな格子においては非常に高い確率で $\lambda_1(L) \approx \mathrm{GH}(L)$ となる.（詳しくは [8] を参照.）格子ベクトル $\mathbf{v}$ に対して、$\|\mathbf{v}\|/\mathrm{GH}(L)$ を $\mathbf{v}$ の**近似係数**と呼ぶ. 近似係数は $\mathbf{v}$ が最短ベクトルと比較してどの程度大きいかの目安になる.

2.3 最短ベクトル問題の求解アルゴリズム

ここでは格子上の最短ベクトルを見つけるアルゴリズム

をまとめる. これらのアルゴリズムは厳密解法と近似解法に分類でき, 互いに補完関係にある. 具体的には, 厳密解法の膨大な計算コストを削減するための前処理として近似解法アルゴリズムを利用する必要がある.

2.3.1 厳密 SVP 解法アルゴリズム

厳密解法アルゴリズムは最短な非零格子ベクトルを見つけるが, その計算コストは非常に大きい. 基本的にこれらのアルゴリズムは短い格子ベクトルを全数探索し, その探索数は格子次元に関して指数関数的である. 代表的なアルゴリズムとして, 以下に説明する列挙法（ENUM）と篩法（Sieve）がある.

2.3.1.1 列挙法（ENUM）: 多項式的空間計算アルゴリズム

列挙法では, 格子 L の基底 $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ と探索上界 $R > 0$ を入力とし, $\|\mathbf{s}\| \leq R$ を満たす格子 L 上の非零なベクトル $\mathbf{s} \in L$ をすべて出力する. 探索上界 R として $\lambda_1(L)$ や $1.05\mathrm{GH}(L)$ の値を設定する. 列挙法では, 射影した後のノルムが探索上界 R 以下のベクトルを列挙する探索木を生成する. より正確には, その探索木の深さは n で, 各 $1 \leq k \leq n+1$ に対し深さ $n+1-k$ のノードはノルムが探索上界 R 以下の射影格子 $\pi_k(L)$ 上のすべてのベクトルで形成される. 特に $\pi_{n+1}(L) = \{\mathbf{0}\}$ より, ノードの根は零ベクトル $\mathbf{0}$ である. また, 深さ $n+1-k$ のノード $\mathbf{u} \in \pi_k(L)$ の親ノードは, 深さ $n-k$ のノード $\pi_{k+1}(\mathbf{u})$ である. この探索木を深さ優先探索し, R を更新して探索範囲を狭めつつ走査する.

2.3.1.2 篩法（Sieve）: 指数関数的空間計算量アルゴリズム

列挙法に比べて, 篩法は漸近的に良い時間計算量を持つが, 格子次元 n に関する指数関数的な空間計算量 $2^{\Theta(n)}$ を持つ. 篩法の最初のアルゴリズムは Ajtai-Kumar-Sivakumar (AKS) [9] によって提案されたランダム篩アルゴリズムで, 非常に高い確率で最短ベクトルを出力する. AKS によるランダム篩アルゴリズムでは, n 次元格子 L に対し, $\lambda_1(L) \leq r \leq O(\lambda_1(L))$ なる r を選び, 原点を中心とする半径 r の球体 S を考える. このとき, Gauss のヒューリスティックから $\#(L \cap S) = 2^{O(n)}$ と見積もれる. 共通領域 $L \cap S$ に含まれる格子ベクトルを全数探索すれば最短な非零ベクトルを見つけることができるが, AKS のアルゴリズムでは共通領域 $L \cap S$ 上のランダムサンプリングを行う. もし, 共通領域 $L \cap S$ 上で一様ランダムにサンプリングすることができれば, サンプリング数を増やせば 1 に非常に近い確率でサンプル元の中に最短ベクトルが含まれる. 共通領域 $L \cap S$ 上で一様ランダムにサンプリングするのは困難であるため, ある程度短い二つの格子ベクトルの差をとることで, $L \cap S$ 上のベクトルを大量に生成する. 確定的アルゴリズムである列挙法の時間計算量が $2^{O(n^2)}$ であるのに対し, 確率的アルゴリズムである篩法の時間計算量は $2^{O(n)}$ である.

2.3.2 近似 SVP 解法アルゴリズム

近似解法アルゴリズムは厳密解法と比べて非常に高速で, 必ずしも最短とは限らないがかなり短い格子ベクトルを見つけることができる.

2.3.2.1 LLL 基底簡約とその変種

最も有名かつ代表的な近似解法は Lenstra-Lenstra-Lovász (LLL) アルゴリズムである [10]. これは**格子基底簡約**アルゴリズムの一つで, 任意の基底から各基底ベクトルが短くかつ互いの基底ベクトルが直交に近い「良い」基底を見つける.（それに対し, 各基底ベクトルが長くかつ互いの基底ベクトルが平行に近い基底を「悪い」基底と呼ぶ.）簡約パラメータ $\frac{1}{4} < \delta < 1$ に対して, 基底 $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ が **δ-LLL 簡約されている**とは, 次の 2 条件を満たすときをいう：(i) **サイズ簡約条件**：基底 $\mathbf{B}$ の Gram-Schmidt 係数に関して, すべての $j < i$ に対し $|\mu_{i,j}| \leq \frac{1}{2}$ を満たす. (ii) **Lovász 条件**：任意の $2 \leq k \leq n$ に対して, 隣り合う基底ベクトルが $\delta\|\mathbf{b}_{k-1}^*\|^2 \leq \|\pi_{k-1}(\mathbf{b}_k)\|^2$ を満たす. LLL 基底簡約アルゴリズムにおいて, 隣り合う基底ベクトル $\mathbf{b}_{k-1}, \mathbf{b}_k$ が Lovász 条件を満たさない場合にその基底ベクトルを交換しながら LLL 簡約基底を見つけていく. LLL 基底簡約アルゴリズムは格子次元 n に関して多項式的な時間計算量を持つ. さらに, LLL 基底簡約アルゴリズムは一次従属なベクトルに対しても適用可能で, その一次従属性を取り除くことができる.

DeepLLL（LLL with deep insertions）は LLL の単純な一般化で, 必ずしも隣り合わない基底ベクトルの交換を行う. 具体的には, 条件 $\|\pi_i(\mathbf{b}_k)\|^2 < \delta\|\mathbf{b}_i^*\|^2$ を満たすとき, 基底ベクトル $\mathbf{b}_k$ を $\mathbf{b}_i$ の前に挿入する：$(\ldots, \mathbf{b}_{i-1}, \mathbf{b}_k, \mathbf{b}_i, \ldots, \mathbf{b}_{k-1}, \mathbf{b}_{k+1}, \ldots)$.

2.3.2.2 BKZ 基底簡約とその変種

格子 L の基底 $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ が **Hermite-Korkine-Zolotarev (HKZ) 簡約されている**とは, その基底がサイズ簡約基底でかつすべての k において $\|\mathbf{b}_k^*\| = \lambda_1(\pi_k(L))$ を満たすときをいう. 基底 $\mathbf{B}$ と $i < j$ に対して, その部分射影格子基底 $(\pi_i(\mathbf{b}_i), \pi_i(\mathbf{b}_{i+1}), \ldots, \pi_i(\mathbf{b}_j))$ を $\mathbf{B}_{[i,j]}$ と表し, その部分射影格子基底で張られる格子を $L_{[i,j]}$ と表す. **Block-Korkine-Zolotarev (BKZ) 簡約**は HKZ の局所版である [11–13]：ブロックサイズ $\beta \geq 2$ に対して, 格子 L の基底 $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ が **β-BKZ 簡約されている**とは, その基底がサイズ簡約基底でかつすべての $1 \leq j \leq n-\beta+1$ において部分射影格子基底 $\mathbf{B}_{[j,j+\beta-1]}$ が HKZ 簡約されているときをいう. 与えらた格子 L の基底に対して, BKZ 基底簡約アルゴリズムは部分射影格子 $L_{[j,j+\beta-1]}$ 上の最短ベクトルを見つける前処理として LLL 基底簡約アルゴリズムを呼び出すことで, β-BKZ 基底を見つけていく. BKZ 基底簡約アルゴリズムとその改良は [14, 15] におけるソフトウェアライブラリとして実装されている. 特に, より大きなブロックサイズ β を入力することで, より短い格子ベクトルを見つけることができる一方で, その処理時間は β に関して指数関数的に長くなる.

BKZ 基底簡約の改良として DeepBKZ 基底簡約アルゴリズムが [16] で提案されている. DeepBKZ 基底簡約アルゴ

リズムでは，BKZ 基底簡約の主サブルーチンである LLL の代わりに DeepLLL 基底簡約を呼び出す．実用的には，DeepBKZ 基底簡約アルゴリズムは従来の BKZ に比べて小さいブロックでもかなり短い格子ベクトルを見つける．

2.4 SVP チャレンジに対する関連研究

高次元 $n \geq 131$ に対しては 2 つの解法， General Sieve Kernel (G6K) [4] と最新のランダムサンプリング簡約の変種 [17] が SVP チャレンジの記録を占めている．前者は現在の最高次元 170 を解くのに使われているが，その近似係数は 1.04690 と報告されており，格子の最短ベクトルの求解には至っていない．さらに [4, Table 2] と [17, Table 5] では近似係数が 1.04 または 1.05 程度のものが多く，どちらの解法も厳密に SVP を解くことができないことがわかる．一方，列挙法 (ENUM) や篩法 (Sieve) などの厳密 SVP アルゴリズムでは最短ベクトルを求めることが可能である．

篩法は理論的にも実用的にも高次元 n では列挙法よりも高速である．しかし，空間計算量については列挙法は n に関して多項式的であるのに対し，篩法は n に関して指数関数的である．実際に，次元 124 と 127 の場合，篩法ではそれぞれ 160GB と 246GB を必要とすることが [4, Table 2] で報告されているため，スケーラビリティの観点から本研究では列挙法の並列化を検討した．

列挙法では次元が大きくなるにつれ探索木のサイズも急激に大きくなるが，探索木の刈り込み手法 [7] を用いることで大幅な高速化が可能である．この列挙法の並列化として，シンプルなものでは探索木の分割統治アプローチがある [18–21]. 他のアプローチとして，大規模並列計算を十分に活かすための入力格子基底のランダマイズが考案されている [20, 21].

この手法では，入力基底からユニモジュラ変換によって同一の格子を生成する多数の基底を生成し，それを複数のプロセスに分配する．各プロセスは格子基底簡約を実行し，あるプロセスが非零最短ベクトルを見つけるまで探索木を走査する．最近の研究 [7] ではランダマイズと extream pruning に基づく共有メモリ並列化システムを提案しており，4-way Intel E7-4890 v2 CPUs (60 コア) により 最大 100 次元までの厳密 SVPd 求解時間を報告している．本研究では基底のランダマイズと刈り込み手法を用いた DeepBKZ と ENUM をもとに厳密 SVP 求解システム MAP-SVP を構成し，さらにアルゴリズム実行中のベクトル共有機能を拡張した．これらの並列機能は次節で紹介する UG を用いて実装を行うことで，大規模並列化を可能にしている．

格子暗号解析では無いが，暗号における大規模並列化に関する研究としては，RSA と ECDSA の暗号解析へのハイパフォーマンスコンピューティングの応用が [22] にまとめられている．

3. Ubiquity Generator (UG): MAP-SVP 並列化のためのソフトウェアツール

Ubiquity Generator (UG) framework [23] は，分枝限定法ソルバ（本稿では *base solver* とよぶ）を，ソルバの外部からソルバが提供する API を利用して並列化する汎用ソフトウェアツールである．UG は C++による基底クラス群で構成されており，「解」や「子問題」等ソルバ間で転送されるオブジェクトは抽象化されており，base solver の変更を可能としている．またメッセージパッシングを実現するコミュニケータ も抽象化されており，利用する並列化ライブラリの変更を可能としている．UG により実体化されるソルバは ug [base solver 名， 並列化ライブラリ名] として表記される．UG は主に最新のアルゴリズムを実装している整数計画（MIP: Mixed Integer Programming) ソルバ SCIP [24] と共に開発されており，コードは SCIP Optimization Suite [25] に含まれている．分散メモリ環境版であるスパコン等で動作する ug[SCIP,MPI] は ParaSCIP として知られている．また，共有メモリ環境版の PC 上で動作する ug[SCIP,C++11-thread/pthrads] は FiberSCIP として知られており最も良くテストされている．重要な点は，分散メモリ環境版および共有メモリ環境版が，アルゴリズム的には同じ動作をするように設計されている点である．このような設計により，SCIP のように 800,000 行にも及ぶ C によるプログラムをスパコンで動作させる際に，そのデバッグを PC 上で行えるようにしている．ParaSCIP は　Oak Ridge National Laboratory の TITAN 上で 80,000 コアを利用した動作実績があり，よく知られているベンチマーク・セットである MIPLIB [26] の未解決問題（最適解が分かっていない問題）の 12 問に対して最適解を求めている [27]. ParaSCIP および FiberSCIP の詳細については Shinano et al. [27, 28] を参照されたい．

SCIP はプラグイン・ベースのソルバであり，特定の問題に対するプラグインを開発することで，特定の問題専用のソルバを開発できる．現在では，このプラグインが ParaSCIP および FiberSCIP にも利用可能となっている．例えば，スタイナー木問題とその派生問題を解くために開発されたプラグインを利用するスタイナー木問題ソルバ SCIP-Jack を UG により並列化した ug [SCIP-Jack, MPI] はスタイナー木問題に対するベンチマーク・セットである SteinLib [29] の未解決問題の 3 問を解いた [30]. また，アルゴリズムおよび並列化性能を向上をした結果, ug [SCIP-Jack, MPI] は 43,000 コアの利用により，さらに 1 問の未解決問題を解いている [31].

分枝限定法による最適化ソルバへの UG による並列化の柔軟性およびスケーラビリティが示されてきた．そのような結果が非分枝限定法ベースのソルバへの UG の適用を促した．UG により並列化されるソルバには 2 種類のプロセス

（または，共有メモリ環境版ではスレッド）が存在する．1 つは `LC` で並列木探索を制御する．もう 1 つは `Solver` で子問題を解く．`LC` の主たる機能は，動的負荷分散である．また，複数のソルバをランダマイズしたパラメタ設定で動作させるような機能 (*racing ramp-up*) も含まれている（詳細は [28] 参照）．非分枝限定法ベースのソルバに対して UG を適用する本質的な問題は， base solver が分枝限定法で動作していることを仮定している点である．本質的には LC を抽象化してカスタマイズ可能な構造とした上で非分枝限定法ソルバを利用できるようにする必要ある．しかし本研究では実験的に分枝限定法に対応するコードを残したまま LC を抽象化し，新しい仮想関数 *parallelDispatch* を追加した．次章以降に，どのように SVP に対するアルゴリズムが UG により並列化されたかを述べる．

4. 大規模並列ソルバ MAP-SVP の開発

この節では SVP 求解ソルバである MAP-SVP の構造を示す．前述した UG により ランダマイズや刈り込み手法を用いて DeepBKZ と ENUM の 2 つの SVP 求解アルゴリズムが並列化できる．これらの手法を用いた動機を 4.1 節で，実際の MAP-SVP の並列システムを 4.2 節以下で説明する．

4.1 ランダマイズ並列アプローチと列挙木の刈り込み

入力基底のランダマイズ並列アプローチは [20] で提案されている．この手法ではランダムに生成したユニモジュラ行列を用いて格子基底を変換し， 複数プロセスで BKZ と ENUM アルゴリズムを独立に並列で最短な非零格子ベクトルが見つかるまで実行する．このランダマイズ並列手法は BKZ アルゴリズムの挙動が入力基底に大きく依存することに基づく．この性質を図 2 で実験的に示す．図 2 は次元 $n = 100, 110, 120$ の 3 つの SVP チャレンジの格子基底をそれぞれ 126 通りにランダマイズし， その後ブロックサイズ $\beta = 30$ の DeepBKZ 基底簡約により 得られた最短な基底ベクトルのノルムを描写したものである．図 2 は 入力基底のランダマイズにより DeepBKZ 簡約後の格子基底にばらつきが生じていることを示しており，特に 100 次元の結果には， GH よりもノルムが小さいベクトルが得られているものも存在している．

またランダマイズ並列化の利点として，*extreme pruning* による ENUM 探索コストの削減がある．extreme pruning は探索木に最短ベクトルが含まれる確率が p となるように確率的に探索木の刈り込みを行うテクニックである．格子基底のランダマイズにより， それぞれのプロセスで独立の確率で探索木の刈り込みが行われると仮定でき，m 個の探索木に対し 少なくとも 1 つの探索木に最短ベクトルが P の確率で含まれるようにするには $p = 1 - \sqrt[m]{1-P}$ とすれば良いと言える．これよりプロセス数 m の増加に伴い， ENUM の探索コストをより大きく削減できることがわかる．

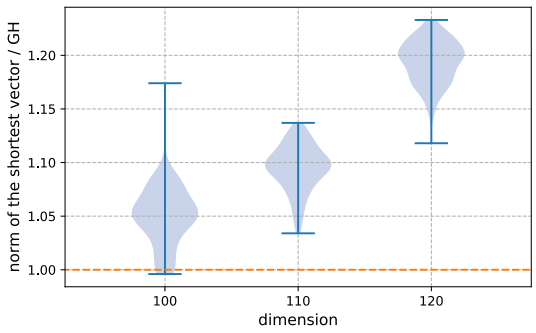

図 2 DeepBKZ 簡約基底の最短ベクトルノルム / $GH(L)$ のバイオリンプロット

Fig. 2 A violin plot of the short vectors of the DeepBKZ reduced lattice basis.

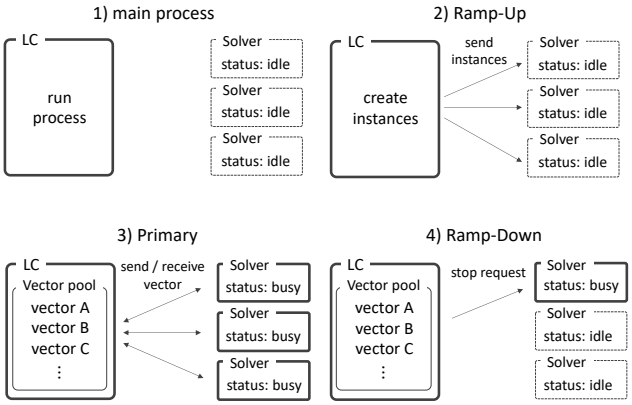

図 3 parallelDispatch の基本フェーズ

Fig. 3 Basic phases of the parallelDispatch

4.2 ParallelDispatch: UG による並列化と 複数 Solver 間の情報共有

DeepBKZ と ENUM のランダマイズ並列システムを実現するためのフレームワークとして UG を用いると共に, 仮想関数 *parallelDispatch* を新たに開発した．UG は管理プロセスである LC と複数の `Solver` から構成される．parallelDispatch では LC が複数の `Solver` にインスタンスを送信し，`Solver` はインスタンス処理中に LC を介して `Solver` 同士が非同期に情報を共有する．parallelDispatch は主に以下の 4 つの実行フェーズから構成される．なお parallelDispatch を用いた並列システムの概要は 4.6 節にまとめておく．

4.2.1 Main process フェーズ

全ての `Solver` は遊休状態で， LC のみが処理を実行しているフェーズ．全体的な前処理および前回の並列計算の結果の収集や 次の並列計算の準備を行う．

4.2.2 Ramp-Up フェーズ

`Solver` がインスタンスを受信して処理を開始するまでのフェーズ．LC は順番に `Solver` にインスタンスを生成し送信する．そのため `Solver` によってはインスタンスの受信が遅れることになる．その待機時間を *start idle time* とする．

4.2.3 Primary フェーズ

全ての Solver が与えられたインスタンスを処理しているフェーズ. Solver は LC との間で， 非同期にベクトルの送受信を行い， LC を介して全ての Solver 間で情報共有を行う. LC は *vector pool* と呼ばれる優先度付きキューで Solver から送信されるベクトルを管理する. 各 Solver は， それぞれが都合の良いタイミングでベクトルの送受信を行うことができ， Solver が受信要求を LC に送ると，LC は vector pool から適切なベクトルを Solver に送信する. LC は複数の Solver からの送受信要求を処理するため，送受信の間にタイムラグが発生する. これを *wait idle time* と呼ぶ.

4.2.4 Ramp-Down フェーズ

少なくとも一つの Solver が遊休状態であるフェーズ. 処理終了が遅い Solver に対しては， LC が処理実行中の Solver に停止要求を送信する. 停止要求を受け取った Solver が即座に処理を終了することで， 計算機全体の遊休時間を短縮させ計算資源を効率的に使用する.

4.3 ベクトル共有を用いた並列 ENUM アルゴリズム

この節では， parallelDispatch によるベクトル共有を用いた並列 ENUM アルゴリズムを提案する. ENUM はノルムがパラメータ R よりも短い非零格子ベクトルを列挙する. ENUM の探索空間は列挙木と呼ばれる， 葉が格子ベクトルに対応する木構造として表現され，走査中に $\|\mathbf{v}^*\| < R$ なるベクトル $\mathbf{v}^*$ を見つけた場合， R を $\|\mathbf{v}^*\|$ に更新することで， 最短ベクトルを列挙木に残しつつ列挙木の刈り込みがされる. 同様に他のプロセスが $\|\mathbf{w}^*\| < R$ なるベクトル $\mathbf{w}^*$ を見つけた場合も，R を $\|\mathbf{w}^*\|$ に更新することで， システム全体での最短ベクトル取得可能性を損なうことなく探索空間を削減できる. これに基づき， MAP-SVP では Solver は最短ベクトルが更新されると LC に送信し， また LC にて最短ベクトルが更新されている場合そのベクトルを受け取り R を更新して探索を続ける. extreme pruning を用いた場合でも同様の探索空間の削減方法を用いることができる.

4.4 ベクトル共有を用いた並列 DeepBKZ アルゴリズム

ENUM と同様にベクトル共有を用いた DeepBKZ アルゴリズムを提案する. DeepBKZ では新しいベクトルを生成するステップ， それの基底への挿入と MLLL [32] による一次従属性の除去のステップ， DeepLLL による簡約処理のステップを繰り返し行う. 新しいベクトル生成ステップでは ある k に対し $\|\pi_k(\mathbf{v})\| = \lambda_1(\pi_k(L))$ となる格子ベクトル $\mathbf{v}$ を生成することを目標とし， 具体的には β-DeepBKZ の場合は射影格子 $L_{[k,k+\beta-1]}$ に対する ENUM により $\|\mathbf{b}_k^*\|$ よりも直交射影 $\pi_k(\mathbf{v})$ が短いベクトル $\mathbf{v}$ を探索する. 一般的に格子ベクトルが短いほど直交射影ベクトルのノルムも短くなるため， 他のプロセスで発見した短いベクトル $\mathbf{v}$ がある k' に対し $\|\mathbf{b}_{k'}^*\| > \|\pi_{k'}(\mathbf{v})\|$ を満たす場合が多く， この場合ベクトル $\mathbf{v}$ を $k = k'$ に対するベクトル生成ステップの代わりとして利用できる. 特に自身の最短ベクトルよりも短いベクトルを他のプロセスから受け取った場合は， $k = 1$ の場合に相当し， 必ず自身の格子基底に挿入される. DeepBKZ を実行している各プロセスは， 短いベクトルを見つけた場合はそのベクトルを LC に送信し， LC はそれを vector pool に格納する. また他のプロセスで見つかった短いベクトルを得るために， ある一定の間隔で受信要求を LC に送信しベクトルを受け取る. この一連の流れにより， DeepBKZ は全プロセスで協調的な実行が可能となる.

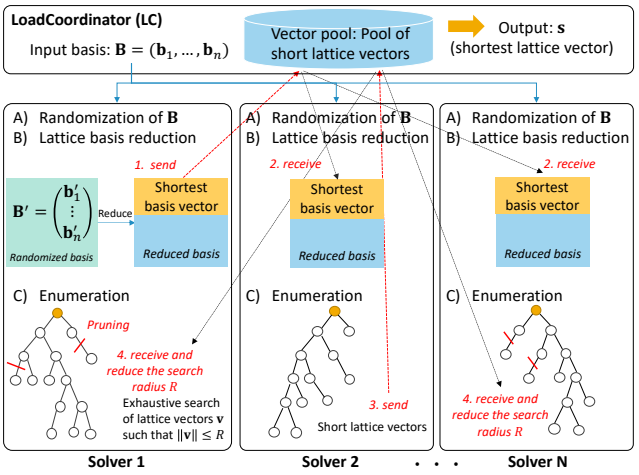

図 4 MAP-SVP の全体像

Fig. 4 An overview of our exact-SVP solving system

4.5 チェックポイントとリスタート

本節では MAP-SVP のチェックポイントとリスタート機能について説明する. DeepBKZ アルゴリズムの場合， 現在の格子基底と処理箇所を表す整数値のみを保存すれば，完全に同じ状態からリスタートできる. ENUM の場合， 列挙木の深さ優先探索は厳密な順序で実行されるため， 列挙木に相当する格子基底と， 前回の探索時の最終ノードの情報のみでリスタートできる. さらに各アルゴリズムは再起動のための追加コストも小さい. このリスタートの簡易さは大規模な計算機が限られた時間でしか使用できない場合が多いことを踏まえると， MAP-SVP の大きな利点である.

4.6 システムの全体像

格子 L の格子基底 $\mathbf{B}$ が与えられた場合. 最短非零ベクトルを見つけるための MAP-SVP の手順を示す (図 4).

(1) Ramp-Up フェーズ: LC は入力基底を全ての Solver に分配する. Solver は受け取った基底をランダマイズすることで， Solver ごとに異なる基底を取得する.

(2) Primary フェーズ: 全ての Solver は， 自身の基底に対して格子基底簡約を行う. より簡約された基底を得るために MAP-SVP では短いベクトルを LC を介して共

有する. 具体的には， 格子基底簡約の際に LC は， 全ての Solver から短い格子ベクトルを収集し， vector pool に格納し，Solver は vector pool からベクトルを受け取り， 自身の基底に挿入する. これにより， 特に最短の基底ベクトルを全ての Solver 間で共有できる. 格子基底簡約後， 全ての Solver は， 簡約基底上のノルムが R よりも短い格子ベクトルを列挙する. 特に，最短の格子ベクトルを見つけるために ENUM アルゴリズムを採用した. 格子基底簡約と同様に， 探索コストを削減するために全ての Solver は現状の最短格子ベクトルを共有する.

(3) Ramp-Down フェーズ: 全 Solver の列挙が終了後，LC は vector pool に含まれる最短格子ベクトルを出力する.

5. 数値実験

本節では， いくつかの数値実験により MAP-SVP の性能と並列化効果を示す. 数値実験を行う上で (1) MPI プロセスのみを使用. (2) 1 コアに 1 プロセスを割り当て， ハイパースレッディング機能は使用しない. (3) LC と Solver は 1 つのプロセスに割り当てる， という設定で計算を行なった. MAP-SVP ではほとんどの通信は LC と Solver 間で行われ，ノード内通信はほとんど行われないため，MPI 並列化のみで実行した. 数値実験で使用した計算機を表 1 にまとめた.

5.1 SVP チャレンジの新記録

Darmstadt 工科大が主催する SVP チャレンジ [3] の $n = 104, 111, 121, 127$ の次元において， MAP-SVP により 既存記録よりも短い格子ベクトルを見つけることに成功した (表 2). なお， SVP チャレンジでは， 同じ次元であってもシード値を変えると別のインスタンスを取得でき，かつ既存の格子ベクトルよりも短い格子ベクトルは， 同じ次元でも記録更新が可能である. MAP-SVP では ENUM の探索木刈り込みに確率的アルゴリズムを用いるが，そのパラーメータは確率 0.95 で最短ベクトルを求められるように選んでいる. 実際， 表 2 の格子ベクトル の近似係数は全て 1 に近いか， 1 以下になっている

特に最も大きな次元である 127 次元での SVP チャレンジについて詳細を記載する. MAP-SVP はこの 127 次元に対し, シード 1, 3 の基底でテストを行い， シード 3 のインスタンスについて， SVP チャレンジの記録を更新するベクトルを 7 回の実行で発見した (表 3). 2 回目以降の実行は，前回の実行のチェックポイントとして格納された格子基底からリスタートしている. この記録の近似係数 0.9757 は，127 以上の次元での記録の中で最も低い値であり，このことは MAP-SVP が SVP を高精度に解くことができていることを示している. またシード 1 では， 先行記録と同じベクトルを並列化により 31 時間という短い時間での算出に成功している. この先行記録は G6K [4] ソルバにより約 14 日を要して見つけられたものである.

SVP チャレンジの 127 次元の新記録は，現状厳密解法ソルバで達成された最短ベクトルの記録の中では， 最高次元の記録であることを強調しておく.

最後に 130 次元 SVP 解読実験について報告する. シード 1 のインスタンスについて 127 次元同様， 複数の計算機により複数回の実験を行なっている (表 4). 103,680 プロセスでの実行により，実時間 (wall time) 24 時間ほどで SVP チャレンジに投稿可能な近似係数 1.05 を下回るベクトルを発見した. また 106 時間程度でノルム 2973 (近似係数 1.01923) のベクトルを取得したが，SVP チャレンジの 130 次元の記録は 2870 (近似係数 0.99413) であり， 記録更新には至っていない. 記録更新に向けて計算を続けている.

5.2 UG の並列効率

図 5 は 100 次元の SVP インスタンスを 1 時間実行したときの，各 Solver が LC からインスタンスを受信するまでの待機時間，情報の送受信中に発生するタイムラグの合計，各 Solver の送受信要求数を描写したものである. (各待機時間の定義は、4.2 節に記載されている). 16〜180 プロセスは CAL A， 1,024 プロセス以上は ITO で実行した. 図 5 が示すように実行プロセス数が増加するほど待機時間タイムラグが総送受信回数に伴って増加するが， その時間は全体の計算時間に比べてはるかに小さい. この結果より MAP-SVP は低いオーバーヘッドでの通信が行われていることがわかる.

また DeepBKZ と ENUM アルゴリズムの特徴により，MAP-SVP はほぼメモリを使用しない. 実際 3 時間実行を行なった MAP-SVP の各プロセスにおける最大メモリ使用量は， 155 次元でも約 0.01 GB であり，次元の増加によるメモリ使用量はほぼ変わらない (図 6). したがって MAP-SVP はより大きな次元の SVP に対してもメモリの小さい大規模計算機で実行可能である.

6. 結論

本稿では parallelDispatch を追加した UG をベースに新しい SVP ソルバ MAP-SVP を提案した. MAP-SVP は SVP 求解のための初の非同期協調分散型システムであり， 通信オーバーヘッドが少なく，チェックポイントや再起動が容易でありメモリ使用量も少ないという特徴を持つ. また数値実験により MAP-SVP を用いて SVP チャレンジ において最高 127 次元の記録の更新に成功した. これは現状厳密解法ソルバで達成された最短ベクトル記録の中では， 最大次元の記録である.

本稿のコードは実験的に開発したものであり， 公開されている UG を用いても実現できない. しかし， 本稿での研究成果を踏まえて， UG 自体を非分枝限定法を扱えるように汎用化し， MAP-SVP をスクラッチから再設計してい

表 1　実験に用いた計算機性能
Table 1　Computing platforms used

Machine	Memory / node	CPU	CPU frequency	# of nodes	# of cores
HLRN IV	384 GB	Intel Xeon Platinum 9242 (CLX-AP)	2.30 GHz	1,042	103,680 (96 × 1,080)
ISM	384 GB	Intel Xeon Gold 6154	3.00 GHz	144	5,184 (36 × 144)
ITO	192 GB	Intel Xeon Gold 6154 (Skylake-SP)	3.00 GHz	128	4,608 (36 × 128)
CAL A	32 GB	Intel(R) Xeon(R) CPU E3-1284L v3	1.80 GHz	45	180 (4 × 45)
CAL B	256 GB	Intel(R) Xeon(R) CPU E5-2640 v3	2.60 GHz	4	64 (16 × 4)
CAL C	256 GB	Intel(R) Xeon(R) CPU E5-2650 v3	2.30 GHz	4	80 (20 × 4)

Table 2　New Solutions For the Hall of Fame in the SVP Challenge [3], Found by MAP-SVP

表 2　MAP-SVP による SVP チャレンジ の新記録

次元	シード	Norm	近似係数	#Process	Total time
104	35	2516	0.97173	120	551 seconds
	85	2520	0.97010	120	214 seconds
	82	2529	0.97719	120	432 seconds
111	29	2597	0.96979	2000	792 seconds
	30	2635	0.98382	2000	541 seconds
	8	2660	0.99467	2000	611 seconds
121	4	2780	0.99706	2304	682 minutes
	2	2809	1.00820	2304	481 minutes
127	3*	2790	0.97573	91,200	147 hours
127	1†	2890	1.01429	9,980	31 hours
130	1†	2935	1.01923	103,680	180 hours

*† ベクトル取得のために, 複数の計算機で複数回 MAP-SVP を実行した. この表には, その中の最大のプロセス数と wall time の合計値の概算を記載している.
† これらの解は新記録ではないが, 既存記録と同じ, もしくはそれに十分に近いものであるため記載した.

Table 3　Execution History for the SVP Challenge of 127-dimensional lattice with seed = 3

表 3　127 次元, シード 3 の SVP チャレンジの実行履歴

try	Norm	近似係数	#Process	Wall Time	Machine
1	3186	1.11435	4,608	6 hours	ITO
2	3186	1.11435	180	11 hours	CAL B, C
3	3037	1.06218	4,608	6 hours	ITO
4	2956	1.03397	4,608	6 hours	ITO
5	2956	1.03397	49,152	12 hours	HLRN IV
6	2922	1.02202	5,184	100 hours	ISM
7	2790	0.97573	91,200	6.3 hours	HLRN IV
Total	2790	0.97573		≈ 147 hours	

る. 将来的に本稿のアルゴリズムを含むより強力なソルバのコードを公開する予定である.

謝辞

本研究は JST，CREST，COI，JSPS 科研費 JP16H01707，JP20H04142，九州大学情報基盤研究開発センター研究用計算機システムの先端的計算科学研究プロジェクト，the Research Campus Modal funded by the German Federal Min-

Table 4　Execution History for the SVP Challenge of 130-dimensional lattice with seed = 1

表 4　130 次元, シード 1 の SVP チャレンジの実行履歴

try	Norm	近似係数	#Process	Wall Time	Machine
1	2994	1.03979	103,680	24 hours	HLRN IV
2	2994	1.03979	4,608	30 hours	ITO
3	2974	1.03275	70,200	10 hours	ITO
4	2974	1.03275	103,584	12 hours	HLRN IV
5	2935	1.01923	103,680	14 hours	HLRN IV
6	2935	1.01923	4,608	36 hours	ITO
7	2935	1.01923	103,680	24 hours	HLRN IV
8	2935	1.01923	4,608	30 hours	ITO
Total	2935	1.01923		≈ 180 hours	

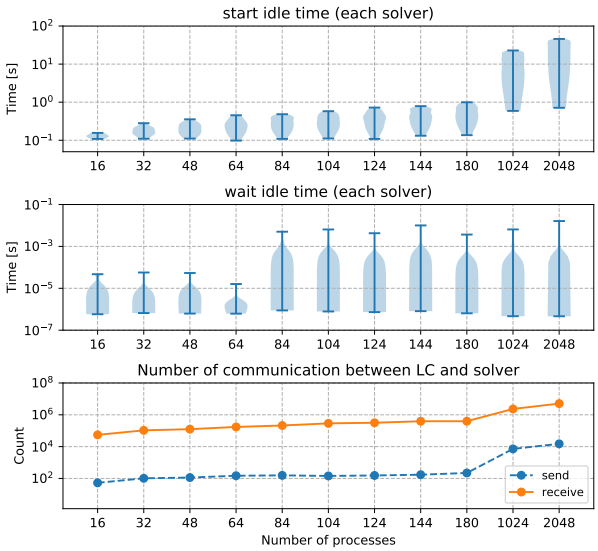

図 5　Solver の待機時間と総送受信要求数
Fig. 5　Idle times and the total number of send and receive request

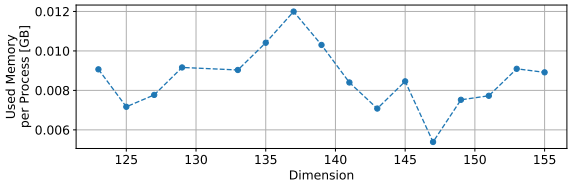

図 6　各次元における MAP-SVP の最大メモリ使用量
Fig. 6　The memory usage in the MAP-SVP for each dimension of SVP

istry of Education and Research (fund number 05M20ZBM), the North-German Supercomputing Alliance (HLRN) の助成を受けたものである. かつ HPC 技術のサポートをしてくださった HLRN IV の HPC スタッフ，特に Matthias Läuter に感謝いたします. また有益な助言とともにいくつかのプログラムを提供いただいた NICT 青野良範 主任研究員に感謝いたします.

参考文献

[1] P. W. Shor, “Algorithms for quantum computation: Discrete logarithms and factoring,” in *Symposium on Foundations of Computer Science (FOCS 1994)*. IEEE, 1994, pp. 124–134.

[2] The National Institute of Standards and Technology (NIST), “Post-quantum cryptography.” [Online]. Available: https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization

[3] M. Schneider, N. Gama, P. Baumann, and L. Nobach, “SVP challenge (2010),” *URL: http://latticechallenge.org/svp-challenge*.

[4] M. Albrecht, L. Ducas, G. Herold, E. Kirshanova, E. W. Postlethwaite, and M. Stevens, “The general sieve kernel and new records in lattice reduction,” in *Advances in Cryptology–EUROCRYPT 2019*, ser. Lecture Notes in Computer Science, vol. 11477. Springer, 2019, pp. 717–746.

[5] N. Tateiwa, Y. Shinano, S. Nakamura, A. Yoshida, S. Kaji, M. Yasuda, and K. Fujisawa, “Massive parallelization for finding shortest lattice vectors based on ubiquity generator framework,” in *the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC20)*, to be held as a virtual conference from 9-19 November 2020.

[6] Y. Shinano, T. Achterberg, T. Berthold, S. Heinz, T. Koch, and M. Winkler, “Solving open mip instances with parascip on supercomputers using up to 80,000 cores,” in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2016, pp. 770–779.

[7] N. Gama, P. Q. Nguyen, and O. Regev, “Lattice enumeration using extreme pruning,” in *Advances in Cryptology–EUROCRYPT 2010*, ser. Lecture Notes in Computer Science, vol. 6110. Springer, 2010, pp. 257–278.

[8] M. Ajtai, “Generating hard instances of lattice problems,” in *Symposium on Theory of Computing (STOC 1996)*. ACM, 1996, pp. 99–108.

[9] M. Ajtai, R. Kumar, and D. Sivakumar, “A sieve algorithm for the shortest lattice vector problem,” in *Symposium on Theory of Computing (STOC 2001)*. ACM, 2001, pp. 601–610.

[10] A. K. Lenstra, H. W. Lenstra, and L. Lovász, “Factoring polynomials with rational coefficients,” *Mathematische Annalen*, vol. 261, no. 4, pp. 515–534, 1982.

[11] C.-P. Schnorr, “A hierarchy of polynomial time lattice basis reduction algorithms,” *Theoretical computer science*, vol. 53, no. 2-3, pp. 201–224, 1987.

[12] ——, *Block Korkin-Zolotarev bases and successive minima*. International Computer Science Institute, 1992.

[13] C.-P. Schnorr and M. Euchner, “Lattice basis reduction: Improved practical algorithms and solving subset sum problems,” *Mathematical programming*, vol. 66, pp. 181–199, 1994.

[14] V. Shoup, “NTL: A Library for doing Number Theory,” available at http://www.shoup.net/ntl/. [Online]. Available: http://www.shoup.net/ntl/

[15] The FPLLL development team, “fplll, a lattice reduction library,” 2016. [Online]. Available: https://github.com/fplll/fplll

[16] J. Yamaguchi and M. Yasuda, “Explicit formula for Gram-Schmidt vectors in LLL with deep insertions and its applications,” in *Number-Theoretic Methods in Cryptology (NuTMiC 2017)*, ser. Lecture Notes in Computer Science, vol. 10737. Springer, 2017, pp. 142–160.

[17] T. Teruya, K. Kashiwabara, and G. Hanaoka, “Fast lattice basis reduction suitable for massive parallelization and its application to the shortest vector problem,” in *Public Key Cryptography (PKC 2018)*, ser. Lecture Notes in Computer Science, vol. 10769. Springer, 2018, pp. 437–460.

[18] Ö. Dagdelen and M. Schneider, “Parallel enumeration of shortest lattice vectors,” in *Euro-Par 2010–Parallel Processing*, ser. Lecture Notes in Computer Science, vol. 6272. Springer, 2010, pp. 211–222.

[19] J. Hermans, M. Schneider, J. Buchmann, F. Vercauteren, and B. Preneel, “Parallel shortest lattice vector enumeration on graphics cards,” in *Progress in Cryptology–AFRICACRYPT 2010*, ser. Lecture Notes in Computer Science, vol. 6055. Springer, 2010, pp. 52–68.

[20] P.-C. Kuo, M. Schneider, Ö. Dagdelen, J. Reichelt, J. Buchmann, C.-M. Cheng, and B.-Y. Yang, “Extreme enumeration on GPU and in clouds,” in *Cryptographic Hardware and Embedded Systems–CHES 2011*, ser. Lecture Notes in Computer Science, vol. 6917. Springer, 2011, pp. 176–191.

[21] M. Burger, C. Bischof, and J. Krämer, “p3Enum: A new parameterizable and shared-memory parallelized shortest vector problem solver,” in *Computational Science–ICCS 2019*, ser. Lecture Notes in Computer Science, vol. 11540. Springer, 2019, pp. 535–542.

[22] A. Joux, “A tutorial on high performance computing applied to cryptanalysis (invited talk),” in *Advances in Cryptology–EUROCRYPT 2012*, ser. Lecture Notes in Computer Science, vol. 7237. Springer, 2012, pp. 1–7.

[23] “UG: Ubiquity Generator framework,” http://ug.zib.de/.

[24] “SCIP: Solving Constraint Integer Programs,” http://scip.zib.de/.

[25] “SCIP Optimization Suite,” https://www.scipopt.org.

[26] “MIPLIB 2017,” 2018, http://miplib.zib.de.

[27] Y. Shinano, T. Achterberg, T. Berthold, S. Heinz, T. Koch, and M. Winkler, “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.

[28] Y. Shinano, S. Heinz, S. Vigerske, and M. Winkler, “Fiberscip—a shared memory parallelization of scip,” *INFORMS Journal on Computing*, vol. 30, no. 1, pp. 11–30, 2018. [Online]. Available: https://doi.org/10.1287/ijoc.2017.0762

[29] T. Koch, A. Martin, and S. Voß, “SteinLib: An updated library on Steiner tree problems in graphs,” in *Steiner Trees in Industries*, D.-Z. Du and X. Cheng, Eds. Kluwer, 2001, pp. 285–325.

[30] G. Gamrath, T. Koch, S. Maher, D. Rehfeldt, and Y. Shinano, “SCIP-Jack—a solver for STP and variants with parallelization extensions,” *Mathematical Programming Computation*, vol. 9, no. 2, pp. 231 – 296, 2017.

[31] Y. Shinano, D. Rehfeldt, and T. 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.

[32] C. C. Sims, *Computation with finitely presented groups*. Cambridge University Press, 1994, vol. 48.