



**RIKKYO UNIVERSITY**

九大IMI共同利用研究集会

"Design and Evaluation for  
New-generation Cryptography"  
「新世代暗号の設計・評価」

# **Introduction to lattice basis reduction and its massive parallelization** (格子基底簡約とその大規模並列化の紹介)

Masaya Yasuda (Rikkyo University)

November 16, 2021

15:40—16:40

# Basics on Lattices

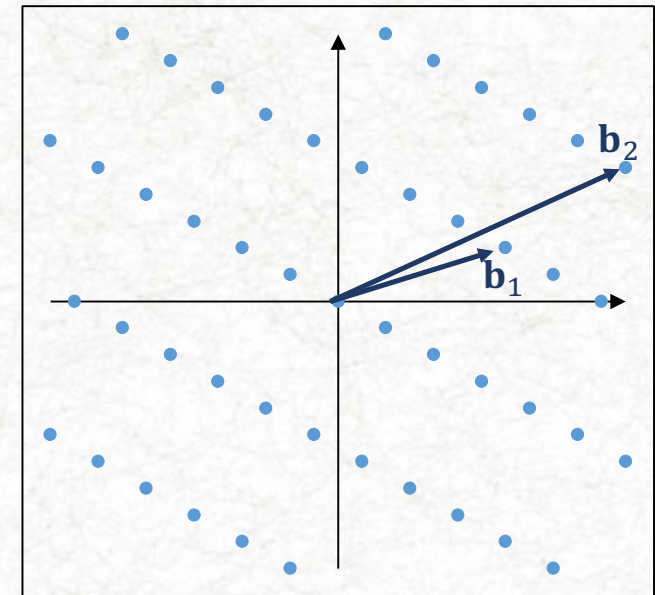
- For linearly independent  $\mathbf{b}_1, \dots, \mathbf{b}_n \in \mathbb{Z}^n$ ,

$$L = \mathcal{L}(\mathbf{b}_1, \dots, \mathbf{b}_n) := \left\{ \sum_{i=1}^n x_i \mathbf{b}_i \mid x_i \in \mathbb{Z} \right\}$$

*Integral combination*

is a (full-rank) **lattice** of dimension  $n$

- $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_n)$ : a **basis** of  $L$ 
  - Regard it as the  $n \times n$  matrix
- Infinitely many bases if  $n \geq 2$ 
  - If  $\mathbf{B}_1$  and  $\mathbf{B}_2$  span the same lattice,
  - then  $\exists \mathbf{V} \in \text{GL}_n(\mathbb{Z})$  such that  $\mathbf{B}_1 = \mathbf{B}_2 \mathbf{V}$
- $\text{vol}(L) = |\det(\mathbf{B})|$ : the **volume** of  $L$ 
  - Independent of the choice of bases



A lattice of dimension  $n = 2$



# Lattices in Cryptography

- **Post-Quantum Cryptography (PQC) Standardization**

- Since 2015, National Institute of Standards and Technology (NIST) has proceeded a standardization project for PQC
  - To standardize quantum-resistant public-key cryptographic algorithms
  - [Post-Quantum Cryptography | CSRC \(nist.gov\)](https://csrc.nist.gov/post-quantum-cryptography)
- In July 2020, NIST selected **7 Finalists** and **8 Alternates**
  - **7 lattice-based schemes** are now in evaluation at the 3<sup>rd</sup> round
    - **5 Finalists (Kyber, NTRU, SABER, Dilithium, Falcon)**
    - **2 Alternates (FrodoKEM, NTRUprime)**

	Signatures		KEM/Encryption		Overall	
	Rd 1	Rd 2	Rd 1	Rd 2	Rd 1	Rd 2
Lattice-based	5	3	21	9	26	12
Code-based	2		17	7	19	7
Multi-variate	7	4	2		9	4
Hash/Symmetric	3	2			3	2
Other	2		5	1	7	1
Total	19	10	45	16	64	26



	Finalists	Alternates
KEMs/Encryption	Kyber NTRU SABER Classic McEliece	Bike FrodoKEM HQC NTRUprime SIKE
Signatures	Dilithium Falcon Rainbow	GeMSS Picnic SPHINCS+

# Lattice Problems

- **Algorithmic problems for lattices**

- **SVP (Shortest Vector Problem)** *Our focus*

- Given a basis  $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_n)$  of a lattice  $L$
- Find a non-zero shortest vector in  $L$

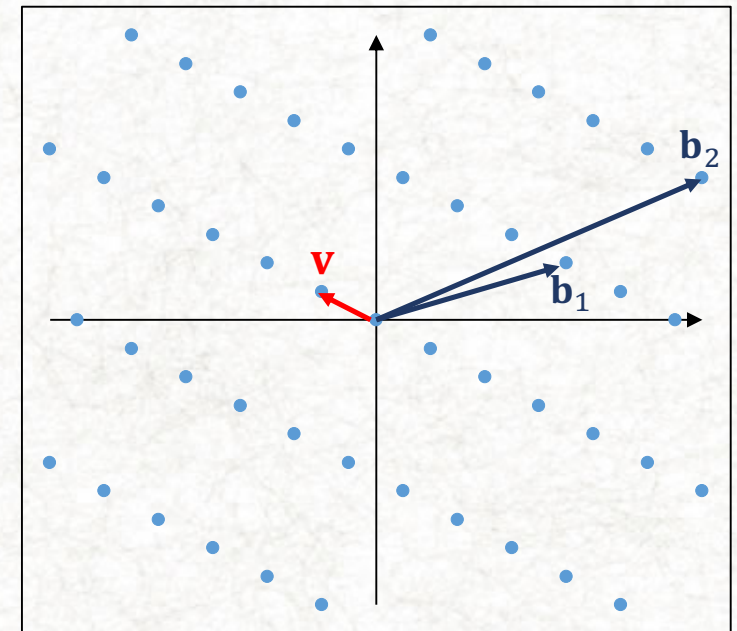
- **CVP (Closest Vector Problem)**

- **LWE (Learning with Errors)**

- **NTRU**, etc.

- **Relationship with cryptography**

- The security of lattice-based cryptography is based on the hardness of lattice problems
- In particular, the hardness of SVP and CVP supports the security of most schemes



SVP in a two-dimensional lattice

- Given linearly independent  $\mathbf{b}_1, \mathbf{b}_2$
- Find a non-zero shortest vector  
 $\mathbf{v} = a_1 \mathbf{b}_1 + a_2 \mathbf{b}_2$  for some  $a_1, a_2 \in \mathbb{Z}$



# Known Results for $\lambda_1(L)$ RIKKYO UNIVERSITY

---

- **The first successive minimum**

- Define  $\lambda_1(L)$  as the length of a non-zero shortest vector in a lattice  $L$
- SVP is the problem that finds  $\mathbf{s} \in L$  such that  $\|\mathbf{s}\| = \lambda_1(L)$

- **Theoretical results**

- Minkowski's convex body theorem implies

$$\lambda_1(L) \leq 2\omega_n^{\frac{1}{n}} \text{vol}(L)^{\frac{1}{n}}$$

for any lattice  $L$  of dimension  $n$  ( $\omega_n$ : the volume of the unit ball in  $\mathbb{R}^n$ )

- **Heuristic results**

- The Gaussian Heuristic implies

$$\lambda_1(L) \approx \omega_n^{\frac{1}{n}} \text{vol}(L)^{\frac{1}{n}} \sim \sqrt{n/2\pi e} \text{vol}(L)^{\frac{1}{n}} =: \text{GH}(L)$$

- The Gaussian Heuristic: The number of vectors in  $L \cap S$  is roughly equal to  $\text{vol}(S)/\text{vol}(L)$  for a measurable set  $S$  in  $\mathbb{R}^n$
- It holds in practice for “random” lattices in high dimensions  $n \geq 50$

# SVP Challenge

- **The Darmstadt SVP challenge**
  - Sample bases  $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_n)$  are presented for dimensions  $40 \leq n \leq 198$
  - Any vector  $\mathbf{v} \in L = \mathcal{L}(\mathbf{B})$  of length  $\|\mathbf{v}\| < 1.05\text{GH}(L) \approx 1.05\lambda_1(L)$  can be submitted to the hall of fame
    - That is, **approximate SVP with factor 1.05**
  - The current highest dimension to be solved is  **$n = 180$**  ([141.pdf \(iacr.org\)](#))
    - It took about **51.6 days** on a server with 4 NVIDIA Turing GPUs with 1.5 TB RAM
    - But the record must be not the shortest (since its approximation factor is about 1.04)



The screenshot shows the SVP Challenge website. The header features the title 'SVP CHALLENGE' over a blue-tinted image of trees. The main content area is divided into sections: 'INTRODUCTION' (describing the challenge), 'PARTICIPATION' (with 'How to participate' and 'How to enter the Hall of Fame' subsections), and 'HALL OF FAME' (a table of top performers). A sidebar on the right contains links for 'SUBMISSION', 'DOWNLOAD' (with a list of lattice dimensions from 40 to 198), and 'LINKS' (including 'Visual Hall of Fame', 'Latticechallenge', 'Ideal Lattice Challenge', 'LWE Challenge', and 'CONTACT').

**INTRODUCTION**

This page presents sample lattices for testing algorithms that solve the shortest vector problem (SVP) in euclidean lattices. The SVP challenge helps assessing the strength of SVP algorithms, and serves to compare different types of algorithms, like sieving and enumeration. The lattices presented here are random lattices in the sense of Goldstein and Mayer.

**PARTICIPATION**

**How to participate:**

You can either

- download a sample lattice on the right side, or
- use the generator online to produce a lattice with (Integer) seeds of your choice, or
- download the generator and install it with an NTL **older than NTL 9.4** (necessary since NTL 9.4 and later versions use a different pseudorandom number generator) to create challenges on your local machine.

**How to enter the Hall of Fame:**

To enter the hall of fame, you have to submit a vector with

- Higher dimension and Euclidean norm less than  $1.05 \cdot \frac{\Gamma(n/2 + 1)^{1/n}}{\sqrt{\pi}} \cdot (\det \mathcal{L})^{1/n}$  (which is an estimation of the length of a shortest vector in the lattice), or
- A shorter vector than a previous one in the same dimension (with possibly different seed)

**Acknowledgment:**

Special thanks to Yuntao Wang and Junpei Yamaguchi for pointing out the change in the NTL pseudorandom generator and to Yuntao Wang for helping with the online version of the generator.

**HALL OF FAME**

Position	Dimension	Euclidean norm	Seed	Contestant	Solution
1	180	3509	0	L. Ducas, M. Stevens, W. van Woerden	vec
2	178	3447	0	L. Ducas, M. Stevens, W. van Woerden	vec
3	176	3487	0	L. Ducas, M. Stevens, W. van Woerden	vec
4	170	3438	0	L. Ducas, M. Stevens, W. van Woerden	vec
5	158	3240	0	Sho Hasegawa, Yuntao Wang, Eiichiro Fujisaki	vec



# Lattice Basis Reduction

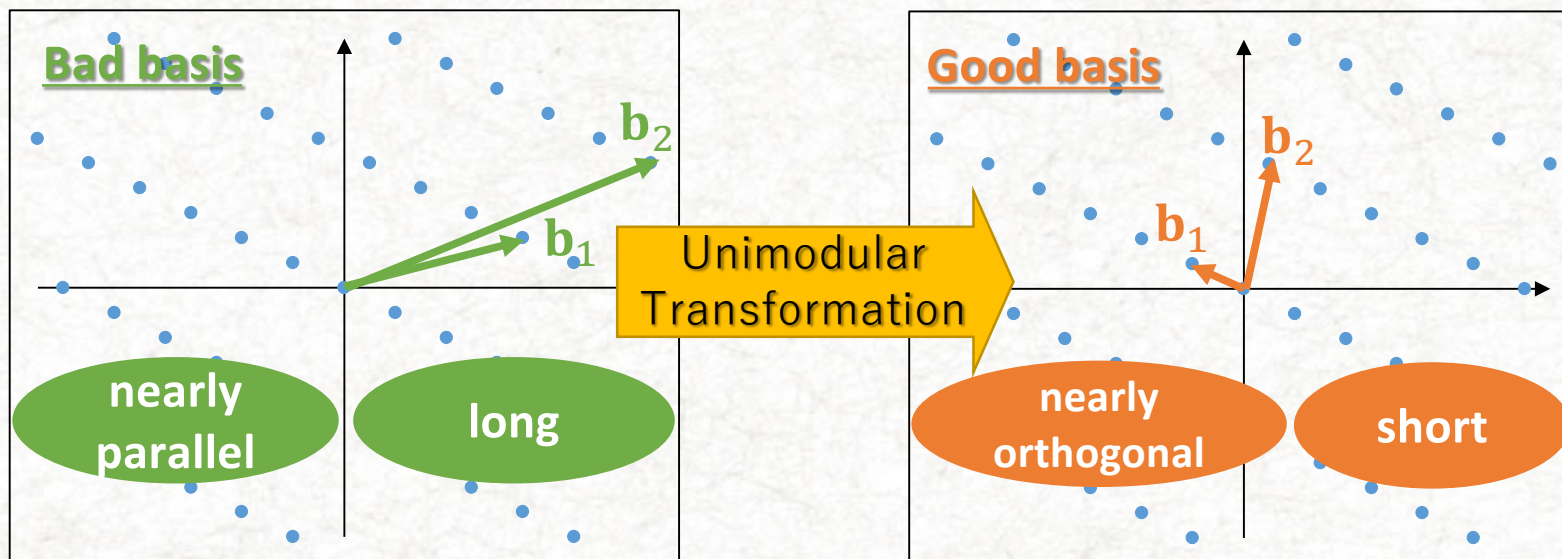
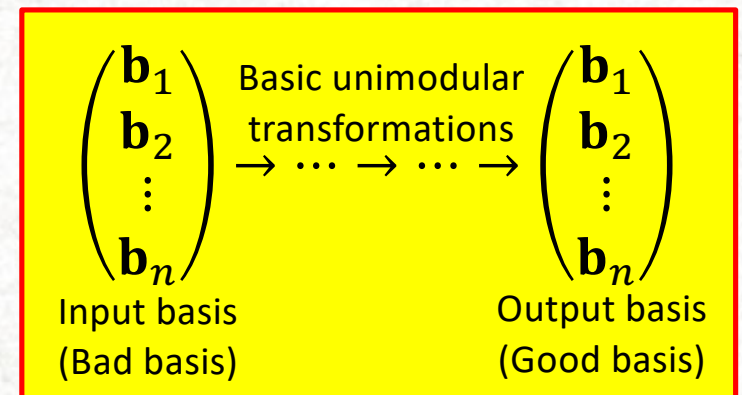
- **Strong tool for solving lattice problems including SVP**

- Find a basis  $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_n)$  with short and nearly orthogonal vectors

- Such a basis is called “good” or “reduced”
- Some basis vectors  $\mathbf{b}_i$ ’s are very short

- Consist of basic unimodular transformations

- ① Multiply by (-1):  $\mathbf{b}_i \leftarrow -\mathbf{b}_i$
- ② Swap  $\mathbf{b}_i$  and  $\mathbf{b}_j$
- ③ Multiply (by integer)-Add:  $\mathbf{b}_i \leftarrow \mathbf{b}_i + a\mathbf{b}_j$  ( $a \in \mathbb{Z}$ )



# LLL (1/3):

## Definition and Properties RIKKYO UNIVERSITY

- **Lenstra-Lenstra-Lovász (LLL)-reduction** [LLL82]
  - $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_n)$  is  **$\delta$ -LLL-reduced** if it satisfies two conditions
    - ① **Size-reduced**:  $|\mu_{ij}| \leq \frac{1}{2}$  for all  $1 \leq j < i \leq n$
    - ② **Lovász' condition**:  $\|\mathbf{b}_k^*\|^2 \geq (\delta - \mu_{k,k-1}^2) \|\mathbf{b}_{k-1}^*\|^2$ 
      - $\frac{1}{4} < \delta < 1$ : reduction parameter (e.g.,  $\delta = 0.99$  for practice)
      - $\mathbf{B}^* = (\mathbf{b}_1^*, \dots, \mathbf{b}_n^*)$ ,  $\mu = (\mu_{ij})$ : Gram-Schmidt information of  $\mathbf{B}$ :

$$\mathbf{b}_1^* = \mathbf{b}_1, \quad \mathbf{b}_i^* = \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{ij} \mathbf{b}_j^*, \quad \mu_{ij} = \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\|\mathbf{b}_j^*\|^2}$$

- Every LLL-reduced basis  $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_n)$  of a lattice  $L$  satisfies
  - $\|\mathbf{b}_1\| \leq \alpha^{\frac{n-1}{2}} \lambda_1(L)$ , where  $\alpha = \frac{4}{4\delta-1} > \frac{4}{3}$
  - $\|\mathbf{b}_1\| \leq \alpha^{\frac{n-1}{4}} \text{vol}(L)^{\frac{1}{n}}$



# LLL (2/3): Basic Algorithm

- It consists of two procedures to find an LLL-reduced basis
  - ① **Size-reduction:**  $\mathbf{b}_k \leftarrow \mathbf{b}_k - q\mathbf{b}_j$  with  $q = \lfloor \mu_{k,j} \rfloor$
  - ② **Swap adjacent vectors:**  $\mathbf{b}_{k-1} \leftrightarrow \mathbf{b}_k$  if they do not satisfy Lovász' condition

**Algorithm: The basic LLL Lenstra et al. (1982)**

**Input:** A basis  $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_n)$  of a lattice  $L$ , and a reduction parameter  $\frac{1}{4} < \delta < 1$

**Output:** A  $\delta$ -LLL-reduced basis  $\mathbf{B}$  of  $L$

- 1: Compute Gram–Schmidt information  $\mu_{i,j}$  and  $\|\mathbf{b}_i^*\|^2$  of the input basis  $\mathbf{B}$
- 2:  $k \leftarrow 2$
- 3: **while**  $k \leq n$  **do**
  - ① 4: Size-reduce  $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_n)$  // At each  $k$ , we recursively change  $\mathbf{b}_k \leftarrow \mathbf{b}_k - \lfloor \mu_{k,j} \rfloor \mathbf{b}_j$  for  $1 \leq j \leq k-1$  (e.g., see Galbraith 2012, Algorithm 24)
  - 5: **if**  $(\mathbf{b}_{k-1}, \mathbf{b}_k)$  satisfies Lovász' condition **then**
  - 6:      $k \leftarrow k + 1$
  - 7: **else**
  - ② 8:     Swap  $\mathbf{b}_k$  with  $\mathbf{b}_{k-1}$ , and update Gram–Schmidt information of  $\mathbf{B}$
  - 9:      $k \leftarrow \max(k-1, 2)$
  - 10: **end if**
  - 11: **end while**

# LLL (3/3): Sage Code



RIKKYO UNIVERSITY

```
1 def GSO(B, n):
2     GS = Matrix(QQ, n)
3     mu = Matrix(QQ, n)
4     for i in range(n):
5         GS[i] = B[i]
6         mu[i, i] = 1
7         for j in range(i):
8             mu[i, j] = B[i].inner_product(GS[j])/GS[j].norm()^2
9             GS[i] -= mu[i, j]*GS[j]
10    return GS, mu
11
12 def LLL(B, n, delta):
13     GS, mu = GSO(B, n)
14     BB = vector(QQ, n)
15     for i in range(n):
16         BB[i] = GS[i].norm()^2
17     k=1
18     while k<=n-1:
19         for j in range(k)[::-1]:
20             if abs(mu[k, j])> 0.50:
21                 q=round(mu[k, j])
22                 B[k] -= q*B[j]
23                 for l in range(j+1):
24                     mu[k, l] -= q*mu[j, l]
25             if BB[k] >= (delta - mu[k, k-1]^2)*BB[k-1]:
26                 k+=1
27             else:
28                 v = B[k-1]; B[k-1]=B[k]; B[k]=v;
29                 GS, mu=GSO(B, n)
30                 for i in range(n):
31                     BB[i] = GS[i].norm()^2
32                 k=max(k-1, 1)
33    return true
```

```
34
35 n = 10; d = 100000
36 B = Matrix(ZZ, n)
37 for i in range(0, n):
38     B[i, i] = 1
39     B[i, 0] = randint(-d, d)
40 print("Input basis")
41 show(B)
42 LLL(B, n, 0.99)
43 print("\nOutput basis")
44 show(B)
```

Please use  
[Sage Cell Server](https://sagemath.org)  
([sagemath.org](https://sagemath.org))



# Enumeration (1/3):

## Basic Idea

- Enumerate all vectors  $\mathbf{s} = \sum v_i \mathbf{b}_i \in \mathcal{L}(\mathbf{B})$  such that  $\|\mathbf{s}\| \leq R$

- $R > 0$ : search radius (e.g.,  $R = 1.05GH(L)$ )

- With Gram-Schmidt information, write

$$\mathbf{s} = \sum_{j=1}^n \left( v_j + \sum_{i=j+1}^n \mu_{ij} v_i \right) \mathbf{b}_j^*$$

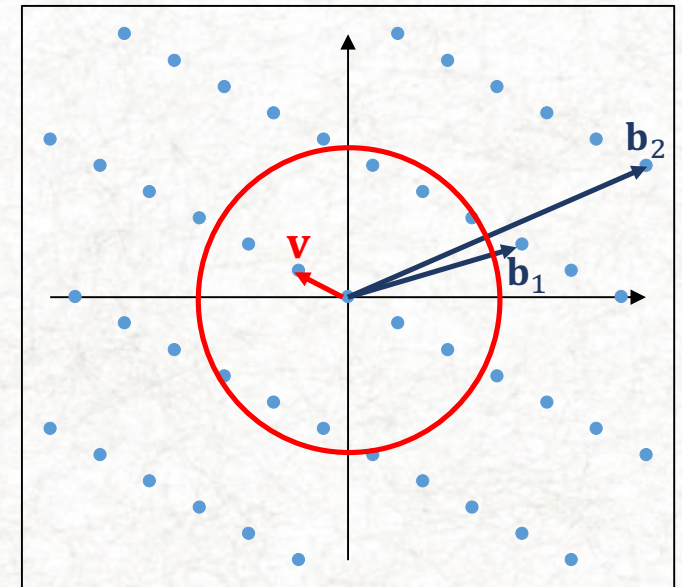
- By the orthogonality of Gram-Schmidt vectors,

$$\|\pi_k(\mathbf{s})\|^2 = \sum_{j=k}^n \left( v_j + \sum_{i=j+1}^n \mu_{ij} v_i \right)^2 \|\mathbf{b}_j^*\|^2$$

for  $1 \leq k \leq n$ , where  $\pi_k$  denotes the projection map to  $\langle \mathbf{b}_k^*, \dots, \mathbf{b}_n^* \rangle_{\mathbb{R}}$

- Consider  $n$  inequalities  $\|\pi_k(\mathbf{s})\|^2 \leq R^2$  for  $1 \leq k \leq n$ :

$$\begin{cases} v_n^2 \leq R^2 / \|\mathbf{b}_n^*\|^2 \\ (v_{n-1} + \mu_{n,n-1} v_n)^2 \leq R^2 - v_n^2 \|\mathbf{b}_n^*\|^2 / \|\mathbf{b}_{n-1}^*\|^2 \\ \vdots \end{cases}$$



# Enumeration (2/3): Basic Algorithm

**Algorithm: The basic Schnorr–Euchner enumeration Schnorr and Euchner (1994)**

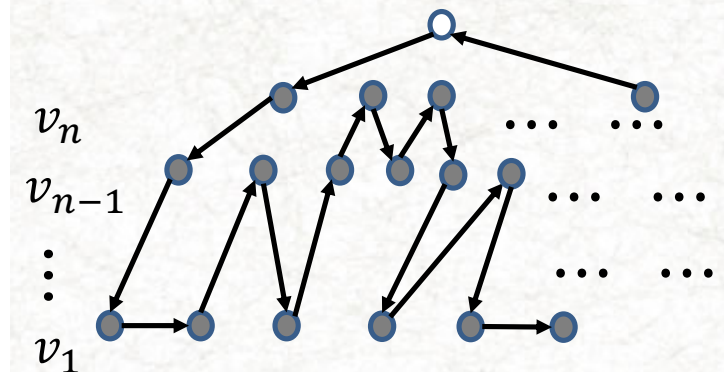
**Input:** A basis  $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_n)$  of a lattice  $L$  and a radius  $R$  with  $\lambda_1(L) \leq R$

**Output:** The shortest non-zero vector  $\mathbf{s} = \sum_{i=1}^n v_i \mathbf{b}_i$  in  $L$

```

1: Compute Gram–Schmidt information  $\mu_{i,j}$  and  $\|\mathbf{b}_i^*\|^2$  of  $\mathbf{B}$ 
2:  $(\rho_1, \dots, \rho_{n+1}) = \mathbf{0}$ ,  $(v_1, \dots, v_n) = (1, 0, \dots, 0)$ ,  $(c_1, \dots, c_n) = \mathbf{0}$ ,  $(w_1, \dots, w_n) = \mathbf{0}$ 
3:  $k = 1$ , last_nonzero = 1 // largest  $i$  for which  $v_i \neq 0$ 
4: while true do
5:    $\rho_k \leftarrow \rho_{k+1} + (v_k - c_k)^2 \cdot \|\mathbf{b}_k^*\|^2$  //  $\rho_k = \|\pi_k(\mathbf{s})\|^2$ 
6:   if  $\rho_k \leq R^2$  then
7:     if  $k = 1$  then  $R^2 \leftarrow \rho_k$ ,  $\mathbf{s} \leftarrow \sum_{i=1}^n v_i \mathbf{b}_i$ ; // update the squared radius
8:     else  $k \leftarrow k - 1$ ,  $c_k \leftarrow -\sum_{i=k+1}^n \mu_{i,k} v_i$ ,  $v_k \leftarrow \lfloor c_k \rfloor$ ,  $w_k \leftarrow 1$ ;
9:   else
10:     $k \leftarrow k + 1$  // going up the tree
11:    if  $k = n + 1$  then return  $\mathbf{s}$ ;
12:    if  $k \geq \text{last\_nonzero}$  then last_nonzero  $\leftarrow k$ ,  $v_k \leftarrow v_k + 1$ ;
13:    else
14:      if  $v_k > c_k$  then  $v_k \leftarrow v_k - w_k$ ; else  $v_k \leftarrow v_k + w_k$ ; // zig-zag search
15:       $w_k \leftarrow w_k + 1$ 
16:    end if
17:  end if
18: end while
  
```

- Enumerate lattice vectors  $\mathbf{s} = \sum v_i \mathbf{b}_i \in L$  such that  $\|\mathbf{s}\| \leq R$
- Built an enumeration tree to find integral combinations  $(v_1, \dots, v_n)$





# Enumeration (3/3): Sage Code



```

1 def GS0(B, n):
2     GS = Matrix(QQ, n)
3     mu = Matrix(QQ, n)
4     for i in range(n):
5         GS[i] = B[i]
6         mu[i, i] = 1
7         for j in range(i):
8             mu[i, j] = B[i].inner_product(GS[j])/GS[j].norm()^2
9             GS[i] -= mu[i, j]*GS[j]
10    return GS, mu
11
12 def ENUM(B, n, R):
13     GS, mu = GS0(B, n)
14     BB = vector(QQ, n)
15     for i in range(n):
16         BB[i] = GS[i].norm()^2
17     sigma = Matrix(QQ, n+1, n)
18     r = vector(ZZ, n+1)
19     rho = vector(QQ, n+1)
20     v = vector(ZZ, n)
21     c = vector(QQ, n)
22     w = vector(ZZ, n)
23     for i in range(n+1):
24         r[i] = i
25     v[0] = 1
26     last_nonzero = 1
27     k = 1
28     while (1):
29         rho[k-1] = rho[k] + (v[k-1] - c[k-1])^2*BB[k-1]
30         if RR(rho[k-1]) <= RR(R):
31             if k==1:
32                 print("Solution found"); return v
33             k = k-1
34             r[k-1] = max(r[k-1], r[k])
35             for i in range(k+1, r[k]+1)[::-1]:
36                 sigma[i-1, k-1] = sigma[i, k-1] + mu[i-1, k-1]*v[i-1]
37             c[k-1] = -sigma[k, k-1]
38             v[k-1] = round(c[k-1])
39             w[k-1] = 1
40         else:
41             k = k+1
42             if k==n+1:
43                 print("No solution"); return false
44             r[k-1] = k
45             if k>=last_nonzero:
46                 last_nonzero = k
47                 v[k-1] = v[k-1] + 1
48             else:
49                 if RR(v[k-1]) > RR(c[k-1]):
50                     v[k-1] = v[k-1] - w[k-1]
51                 else:
52                     v[k-1] = v[k-1] + w[k-1]
53                 w[k-1] = w[k-1] + 1

```

```

55 #Main
56 n = 20
57 B = random_matrix(ZZ, n, x=0, y = 30)
58 B.LLL()
59 print("LLL-reduced basis =%n", B)
60 R = 0.99*RR(B[0].norm()^2)
61 while (1):
62     v = vector(ZZ, n)
63     v = ENUM(B, n, R)
64     if v != false:
65         vec = v[0]*B[0]
66         for i in range(1, n):
67             vec += v[i]*B[i]
68         R = 0.99*RR(vec.norm()^2)
69         print("Norm=", RR(vec.norm()), ", Vector=", vec)
70     else:
71         break
72 print("End")

```

# BKZ (1/3):

## Definition and Properties RIKKYO UNIVERSITY

- **Block Korkine-Zolotarev (BKZ)-reduction**

- A blockwise generalization of LLL with blocksize  $\beta$
- $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_n)$  is  **$\beta$ -BKZ-reduced** if it satisfies two conditions
  - ① It is size-reduced (same as LLL)
  - ② The  $k$ -th Gram-Schmidt vector  $\mathbf{b}_k^*$  is shortest in  $L_{[k, \ell]}$  with  $\ell = \min(k + \beta - 1, n)$  for all  $1 \leq k < n$

$$\begin{array}{llllll}
 L_{[1, \beta]} : & \mathbf{b}_1 & \cdots & \cdots & \mathbf{b}_\beta & \\
 L_{[2, \beta+1]} : & & \pi_2(\mathbf{b}_2) & \cdots & \cdots & \pi_2(\mathbf{b}_{\beta+1}) \\
 \vdots & & & \ddots & & \ddots \\
 L_{[n-\beta+1, n]} : & & & \pi_{n-\beta+1}(\mathbf{b}_{n-\beta+1}) & \cdots & \cdots \pi_{n-\beta+1}(\mathbf{b}_n)
 \end{array}$$

- Every  $\beta$ -BKZ-reduced basis  $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_n)$  of a lattice  $L$  satisfies

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

- $\gamma_\beta$ : Hermite's constant of dimension  $\beta$ , i.e.,  $\gamma_\beta = \sup_L \lambda_1(L)^2 / \text{vol}(L)^{2/n}$
- As  $\beta$  increases,  $\gamma_\beta^{1/(\beta-1)}$  decreases and thus  $\mathbf{b}_1$  can be shorter



# BKZ (2/3): Basic Algorithm

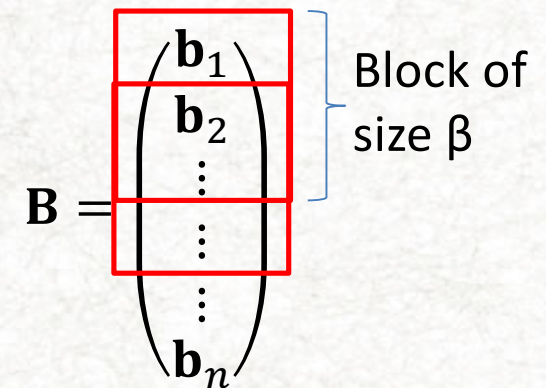
- It consists of LLL and ENUM:
  - Call ENUM to find a non-zero shortest vector in  $L_{[k, \ell]}$
  - Call LLL to reduce a projected block basis of  $L_{[k, \ell]}$

## Algorithm: The basic BKZ Schnorr and Euchner (1994)

**Input:** A basis  $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_n)$  of a lattice  $L$ , a blocksize  $2 \leq \beta \leq n$ , and a reduction parameter  $\frac{1}{4} < \delta < 1$  of LLL

**Output:** A  $\beta$ -DeepBKZ-reduced basis  $\mathbf{B}$  of  $L$

```
1:  $\mathbf{B} \leftarrow \text{LLL}(\mathbf{B}, \delta)$  // Compute  $\mu_{i,j}$  and  $\|\mathbf{b}_j^*\|^2$  of the new basis  $\mathbf{B}$  together
2:  $z \leftarrow 0, j \leftarrow 0$ 
3: while  $z < n - 1$  do
4:    $j \leftarrow (j \bmod (n - 1)) + 1, k \leftarrow \min(j + \beta - 1, n), h \leftarrow \min(k + 1, n)$ 
5:   Find  $\mathbf{v} \in L$  such that  $\|\pi_j(\mathbf{v})\| = \lambda_1(L_{[j,k]})$  by enumeration or sieve
6:   if  $\|\pi_j(\mathbf{v})\|^2 < \|\mathbf{b}_j^*\|^2$  then
7:      $z \leftarrow 0$  and call  $\text{LLL}((\mathbf{b}_1, \dots, \mathbf{b}_{j-1}, \mathbf{v}, \mathbf{b}_j, \dots, \mathbf{b}_h), \delta)$  // Insert  $\mathbf{v} \in L$  and
       remove the linear dependency to obtain a new basis
8:   else
9:      $z \leftarrow z + 1$  and call  $\text{LLL}((\mathbf{b}_1, \dots, \mathbf{b}_h), \delta)$ 
10:  end if
11: end while
```



✂ As reference,  
please look at  
[BKZ-60 – YouTube](#)  
by Martin Albrecht

# BKZ (3/3): Sage Code

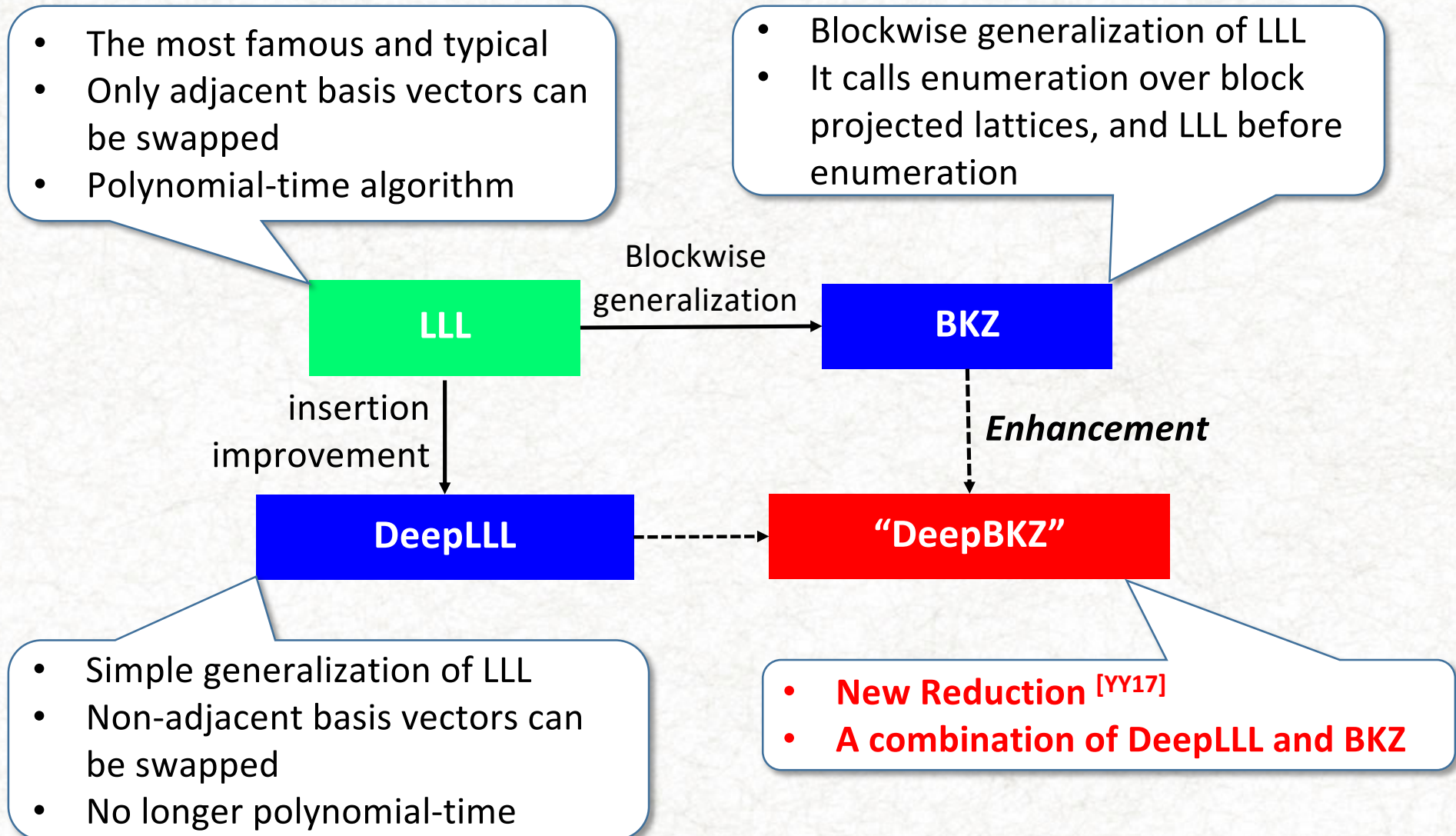


```
12 def ENUM(B, n, R, g, h):
13     BB, U = GSO(B, n)
14     Bnn = vector(QQ, n)
15     for i in range(n):
16         Bnn[i] = BB[i].norm()^2
17     BB, U = GSO(B, n)
18     sigma = Matrix(QQ, n+1, n)
19     r = vector(ZZ, n+1)
20     rho = vector(QQ, n+1)
21     v = vector(ZZ, n)
22     c = vector(QQ, n)
23     w = vector(ZZ, n)
24     for i in range(n+1):
25         r[i] = i
26     v[g] = 1
27     last_nonzero = 1
28     k = g + 1
29     flag = 0
30     v1 = vector(ZZ, n)
31     while (1):
32         rho[k-1] = rho[k] + (v[k-1] - c[k-1])^2*Bnn[k-1]
33         if rho[k-1] <= R:
34             if k==g+1:
35                 R = 0.99*rho[k-1]
36                 flag += 1
37                 for i in range(n):
38                     v1[i] = v[i]
39             k = k-1
40             r[k-1] = max(r[k-1], r[k])
41             for i in range(k+1, r[k]+1)[::-1]:
42                 sigma[i-1, k-1] = sigma[i, k-1] + U[i-1, k-1]*v[i-1]
43             c[k-1] = -sigma[k, k-1]
44             v[k-1] = round(c[k-1])
45             w[k-1] = 1
46         else:
47             k = k+1
48             if k==h+1:
49                 if flag == 0:
50                     return False
51                 else:
52                     vv = v1[g]*B[g]
53                     for i in range(g+1, h+1):
54                         vv += v1[i]*B[i]
55                     return vv
56             r[k-1] = k
57             if k>=last_nonzero:
58                 last_nonzero = k
59                 v[k-1] = v[k-1] + 1
60             else:
61                 if v[k-1] > c[k-1]:
62                     v[k-1] = v[k-1] - w[k-1]
63                 else:
64                     v[k-1] = v[k-1] + w[k-1]
65                 w[k-1] = w[k-1] + 1
```

```
67 def BKZ(B, n, block):
68     B.LLL()
69     BB, U = GSO(B, n)
70     Bnn = vector(QQ, n)
71     for i in range(n):
72         Bnn[i] = BB[i].norm()^2
73     z = 0
74     k = -1
75     while z < n-1:
76         k = lift(mod(k+1, n-2))
77         l = min(k+block-1, n-1)
78         h = min(l+1, n-1)
79         print("(k, l, h) = ", k, l, h)
80
81     R = 0.99*Bnn[k]
82     v = 0
83     v = ENUM(B, n, R, k, l)
84     if v != 0:
85         z = 0
86         C = Matrix(ZZ, h+1, n)
87         for i in range(k):
88             C[i] = B[i]
89         C[k] = v
90         for i in range(k+1, h+1):
91             C[i] = B[i-1]
92         C = C.LLL()
93         for i in range(1, h+1):
94             B[i-1] = C[i]
95         BB, U = GSO(B, n)
96         Bnn = vector(QQ, n)
97         for i in range(n):
98             Bnn[i] = BB[i].norm()^2
99     else:
100         z += 1
101         B = B.LLL()
102         BB, U = GSO(B, n)
103         Bnn = vector(QQ, n)
104         for i in range(n):
105             Bnn[i] = BB[i].norm()^2
106
107     n = 20; d = 1000000
108     B = Matrix(ZZ, n)
109     for i in range(0, n):
110         B[i, i] = 1
111         B[i, 0] = randint(-d, d)
112     show(B)
113     B = B.LLL()
114     BKZ(B, n, 10)
115     show(B)
```



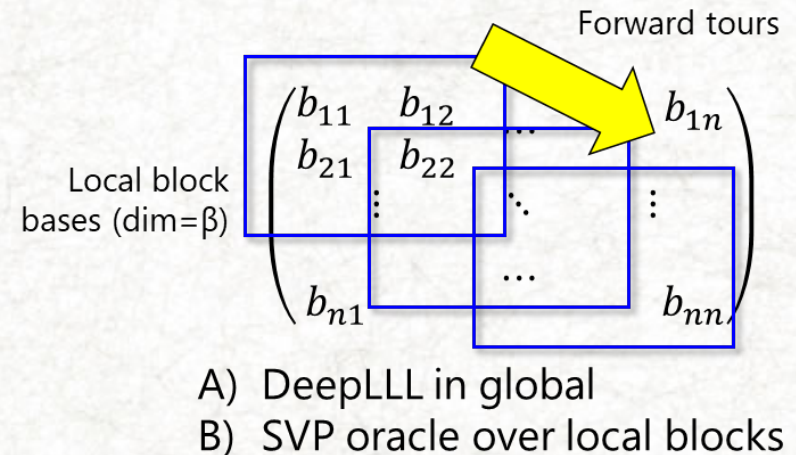
# DeepBKZ (1/6): New Reduction



# DeepBKZ (2/6): Basic Construction

- **Enhancement of BKZ**

- Call **DeepLLL** as a subroutine in BKZ
  - DeepLLL is a straightforward generalization of LLL
  - It is called before every SVP oracle over a  $\beta$ -dimensional lattice



LLL	DeepLLL
Only adjacent basis vectors are swapped	Non-adjacent basis vectors can be changed $\Rightarrow$ shorter basis vectors can be found
$\mathbf{B} \leftarrow (\mathbf{b}_1, \dots, \mathbf{b}_{i+1}, \mathbf{b}_i, \dots, \mathbf{b}_n)$ <p>New basis</p> <p>swap</p>	$\mathbf{B} \leftarrow (\mathbf{b}_1, \dots, \mathbf{b}_k, \mathbf{b}_i, \dots, \mathbf{b}_{k-1}, \mathbf{b}_{k+1}, \dots, \mathbf{b}_n)$ <p>deep insertion</p>

- **Features**

- Even small  $\beta$  can find a very short lattice vector
- DeepLLL is somewhat costly
- But for  $\beta \geq 30$ , SVP-calls are dominant and the cost is same as BKZ



# DeepBKZ (3/6): Gram-Schmidt Formula

- **Complex basis transformation (general form)**

–  $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_n) \rightarrow \mathbf{C} = (\mathbf{b}_1, \dots, \mathbf{b}_{k-1}, \mathbf{v}, \mathbf{b}_k, \dots, \mathbf{b}_{n-1})$

*insert*

•  $\mathbf{v} = \sum_{i=1}^n x_i \mathbf{b}_i = \sum_{i=1}^n v_i \mathbf{b}_i^*$  for some  $x_i \in \mathbb{Z}$  with  $x_n = \pm 1$

- **Gram-Schmidt formula in DeepLLL<sup>[YY+15, YY17]</sup>**

– This enables to make DeepLLL practical like LLL

Proposition: Gram-Schmidt orthogonalization  $[\mathbf{c}_1^*, \dots, \mathbf{c}_n^*]$  of  $\mathbf{C}$

Set  $m = \max \{k \leq i \leq n \mid v_i \neq 0\}$ . Then we have

$$\mathbf{c}_j^* = \begin{cases} \sum_{i=k}^m v_i \mathbf{b}_i^* & \text{for } j = k, \\ \frac{D_j}{D_{j-1}} \mathbf{b}_{j-1}^* - \sum_{i=j}^m \frac{v_i v_{j-1} \|\mathbf{b}_{j-1}^*\|^2}{D_{j-1}} \mathbf{b}_i^* & \text{for } k+1 \leq j \leq m+1, \\ \mathbf{b}_{j-1}^* & \text{for } m+2 \leq j \leq n+1, \end{cases}$$

where  $D_\ell = \sum_{i=\ell}^m v_i^2 \|\mathbf{b}_i^*\|^2$  for  $1 \leq \ell \leq m$ . In particular,  $\mathbf{c}_{m+1}^* = \mathbf{0}$ .

For  $k+1 \leq j \leq m$ , we have

$$\|\mathbf{c}_j^*\|^2 = \frac{D_j}{D_{j-1}} \|\mathbf{b}_{j-1}^*\|^2.$$

# DeepBKZ (4/6): Properties of Reduction

- $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_n)$ : DeepBKZ-reduced

①  $\delta$ -DeepLLL-reduced ( $1/4 < \delta < 1$ )

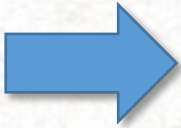
- Size-reduced
- $\delta \|\mathbf{b}_i^*\|^2 \leq \|\pi_i(\mathbf{b}_k)\|^2$  for all  $i < k$

②  $\beta$ -BKZ-reduced ( $2 \leq \beta \leq n$ )

- $\|\mathbf{b}_i^*\| = \lambda_1(\pi_i(L))$  for all  $1 \leq i \leq n$   
( $\pi_i$  is the orthogonal projection)

- Then we have

$$\delta \|\mathbf{b}_1\|^2 \leq \|\mathbf{b}_{\beta+1}\|^2 \leq \|\mathbf{b}_{\beta+1}^*\|^2 + \frac{1}{4} \sum_{j=1}^{\beta} \|\mathbf{b}_j^*\|^2$$


$$\left(\delta - \frac{1}{4}\right) \frac{\|\mathbf{b}_1\|^2}{\|\mathbf{b}_{\beta+1}^*\|^2} \leq 1 + \frac{1}{4} \sum_{j=2}^{\beta} \frac{\|\mathbf{b}_j^*\|^2}{\|\mathbf{b}_{\beta+1}^*\|^2} \leq 1 + \frac{C_{\beta}}{4},$$

where  $C_{\beta} = \max \sum_{i=1}^{\beta-1} \frac{\|\mathbf{b}_i^*\|^2}{\|\mathbf{b}_{\beta}^*\|^2}$  over all HKZ-reduced  $(\mathbf{b}_1, \dots, \mathbf{b}_{\beta})$



# DeepBKZ (5/6): Provable Output Quality

- **Lemma**<sup>[YNY20]</sup>

- Every DeepBKZ-reduced basis  $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_n)$  satisfies

$$\frac{\|\mathbf{b}_1\|^2}{\|\mathbf{b}_{i\beta+1}^*\|^2} \leq \alpha \left(1 + \frac{C_\beta}{4}\right) \left\{1 + \frac{\alpha(1 + C_\beta)}{4}\right\}^{i-1}$$

for  $i \geq 1$ , where  $\alpha = \frac{4}{4\delta-1} > \frac{4}{3}$

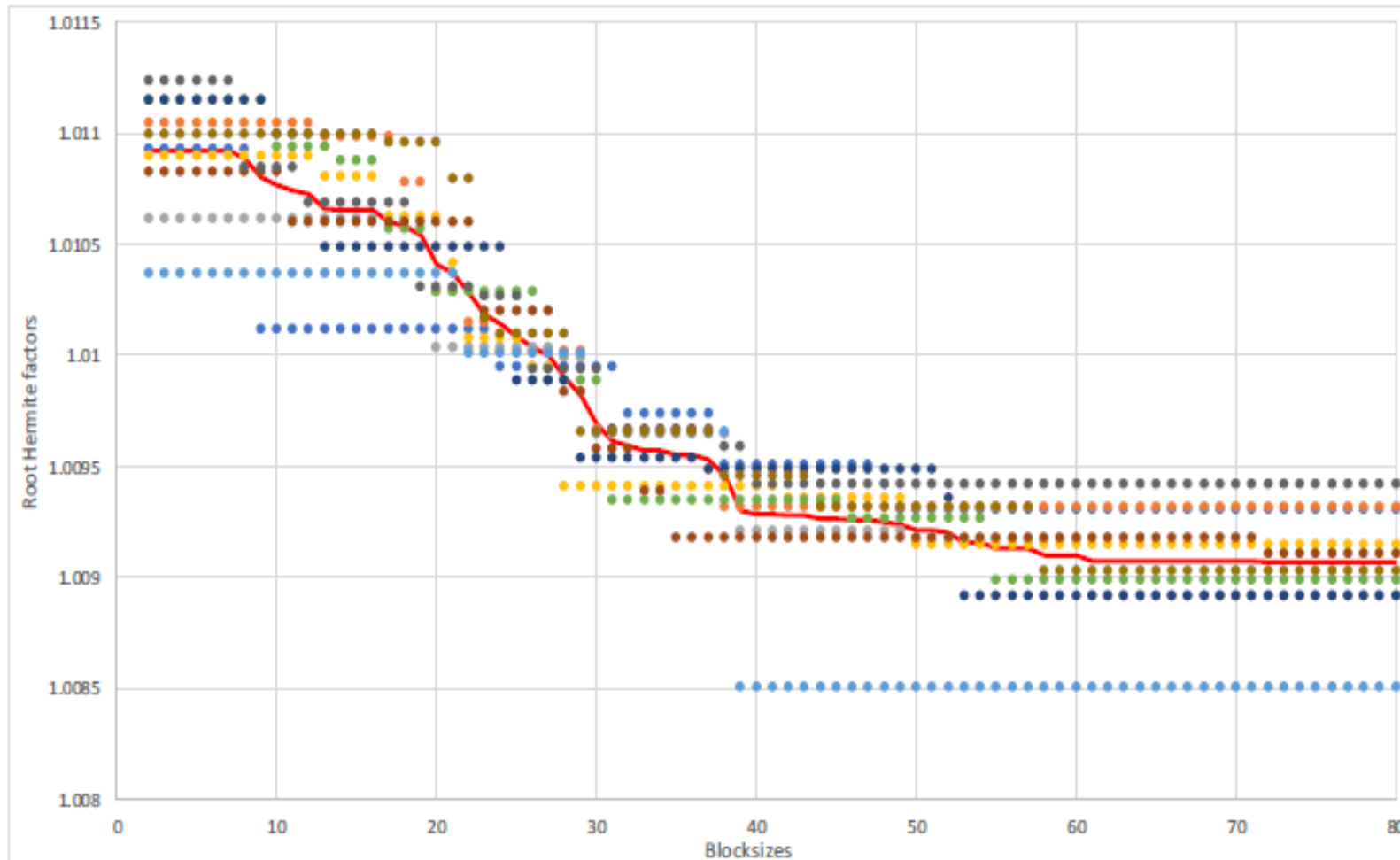
- **Theorem**<sup>[YNY20]</sup>

- $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_n)$ :  $(\delta, \beta)$ -DeepBKZ-reduced basis of  $L$
- Assume  $n$  is divisible by  $\beta$  with  $p = \frac{n}{\beta} \geq 2$
- Then we have

$$\frac{\|\mathbf{b}_1\|}{\text{vol}(L)^{1/n}} \leq \sqrt{\gamma_\beta} \left\{ \alpha \left(1 + \frac{C_\beta}{4}\right) \right\}^{\frac{\beta(p-1)}{2n}} \left\{ 1 + \frac{\alpha(1 + C_\beta)}{4} \right\}^{\frac{\beta(p-1)(p-2)}{4n}}$$

where  $\gamma_\beta$  is Hermite's constant of dimension  $\beta$

# DeepBKZ (6/6) : Practical Output Quality



**Fig. 1** The root Hermite factor of DeepBKZ with blocksizes  $2 \leq \beta \leq 80$  for the SVP challenge in dimension  $n = 115$  with seeds 0–9 (Each dot denotes the root Hermite factor for some seed, and the polygonal line denotes the average.)



# New SVP Solutions by (Parallel) DeepBKZ

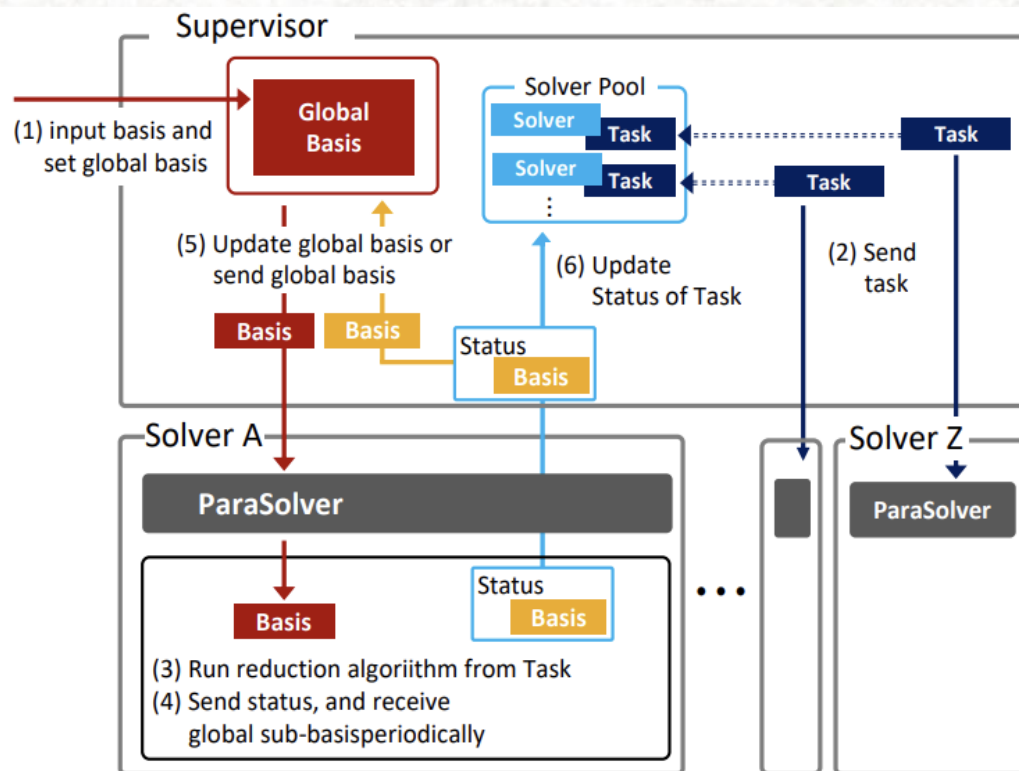
- **DeepBKZ found many new solutions for the SVP challenge**
  - In most dimensions up to  $n = 128$
  - We used block sizes  $\beta = 30\text{--}45$
  - Our solutions are the shortest or very close to it
    - Since their approximation factors are close to 1.0 (0.98470 for  $n = 128$ )
  - For  $n = 128$ , it took about 57.5 hours by massive parallel computation using 24,576 cores

67	130	3025	0	Kenji Kashiwabara and Masaharu Fukase	<a href="#">vec</a>	Other	2013-11-15	1.04787
68	129	2818	1	Yuga Miyagi and Eiichiro Fujisaki	<a href="#">vec</a>	Sieving	2019-04-11	0.98161
69	129	2855	0	Yuga Miyagi, Tomohiro Sekiguchi, Eiichiro Fujisaki	<a href="#">vec</a>	Sieving	2019-03-26	0.99172
70	129	2875	0	Martin Albrecht, Leo Ducas, Gottfried Herold, Elena Kirshanova, Eamonn Postlethwaite, Marc Stevens	<a href="#">vec</a>	Sieving	2018-08-30	0.99878
71	129	2988	0	Martin R. Albrecht, Leo Ducas, Gottfried Herold, Elena Kirshanova, Eamonn Postlethwaite and Marc Ste	<a href="#">vec</a>	Sieving	2018-08-30	1.03813
72	128	2812	1	N. Tateiwa, Y. Shinano, K. Yamamura, A. Yoshida, S. Kajli, M. <b>Yasuda</b> , K. Fujisawa	<a href="#">vec</a>	BKZ	2021-10-16	0.98470
73	128	2882	0	Kenji KASHIWABARA and Tadanori TERUYA	<a href="#">vec</a>	Other	2018-07-9	1.00477
74	128	2948	0	Kenji KASHIWABARA and Tadanori TERUYA	<a href="#">vec</a>	Other	2018-06-20	1.02755
75	128	2974	0	Kenji KASHIWABARA and Tadanori TERUYA	<a href="#">vec</a>	Other	2018-05-9	1.03665
76	128	2984	0	Kenji Kashiwabara and Masaharu fukase	<a href="#">vec</a>	Other	2013-09-23	1.04017
77	128	2992	0	Kenji Kashiwabara and Masaharu Fukase	<a href="#">vec</a>	Other	2013-09-19	1.04313
78	127	2790	3	N. Tateiwa, Y. Shinano, A. Yoshida, S. Nakamura, S. Kajli, M. <b>Yasuda</b> , Y. Aono, K. Fujisawa	<a href="#">vec</a>	ENUM,BKZ,Other	2020-06-7	0.97573
79	127	2890	1	Yuga Miyagi and Eiichiro Fujisaki	<a href="#">vec</a>	Sieving	2019-04-11	1.01429
80	127	2898	0	Yuga Miyagi, Tomohiro Sekiguchi, Eiichiro Fujisaki	<a href="#">vec</a>	Sieving	2019-03-25	1.01626
81	127	2932	2	Junpei Yamaguchi, Masaya <b>Yasuda</b> and Takuya Hayashi	<a href="#">vec</a>	Other	2018-01-12	1.02804
82	126	2812	0	Tadanori TERUYA	<a href="#">vec</a>	Sieving,Other	2019-04-2	0.99052
83	126	2855	0	Yoshinori Aono and Phong Nguyen	<a href="#">vec</a>	ENUM,BKZ	2014-09-9	1.00556
84	126	2897	0	Kenji KASHIWABARA and Tadanori TERUYA	<a href="#">vec</a>	Other	2014-08-27	1.02051
85	126	2906	0	Yoshinori Aono	<a href="#">vec</a>	ENUM,BKZ	2014-07-14	1.02357
86	126	2944	0	Kenji Kashiwabara and Masaharu Fukase	<a href="#">vec</a>	Other	2013-09-4	1.03679
87	126	2969	42	Yuanmi Chen and Phong Nguyen	<a href="#">vec</a>	ENUM,BKZ	2013-04-12	1.04356
88	125	2806	3	Jim Johnson	<a href="#">vec</a>	Sieving	2021-10-22	0.99077
89	125	2834	0	Tadanori TERUYA	<a href="#">vec</a>	Sieving,Other	2019-04-2	1.00341
90	125	2907	3	Junpei Yamaguchi, Masaya <b>Yasuda</b> and Takuya Hayashi	<a href="#">vec</a>	Other	2017-11-20	1.02649
91	125	2922	8	Junpei Yamaguchi, Masaya <b>Yasuda</b> and Takuya Hayashi	<a href="#">vec</a>	ENUM,Other	2017-10-15	1.03203

# Massive Parallelization of DeepBKZ (1/5)

- **Parallel sharing DeepBKZ**

- Distributed and asynchronous system using randomization and DeepBKZ
- Using CMAP-LAP<sup>[TS+21]</sup>, a general framework for lattice algorithms



## Supervisor-Solvers Style

- Every solver runs DeepBKZ on a randomized basis independently
- Supervisor collects short basis vectors from solvers, and distributes them to solvers
- **Every solver uses distributed vectors to accelerate its reduction process**  
(See [TS+20] for sharing a shortest basis vector)

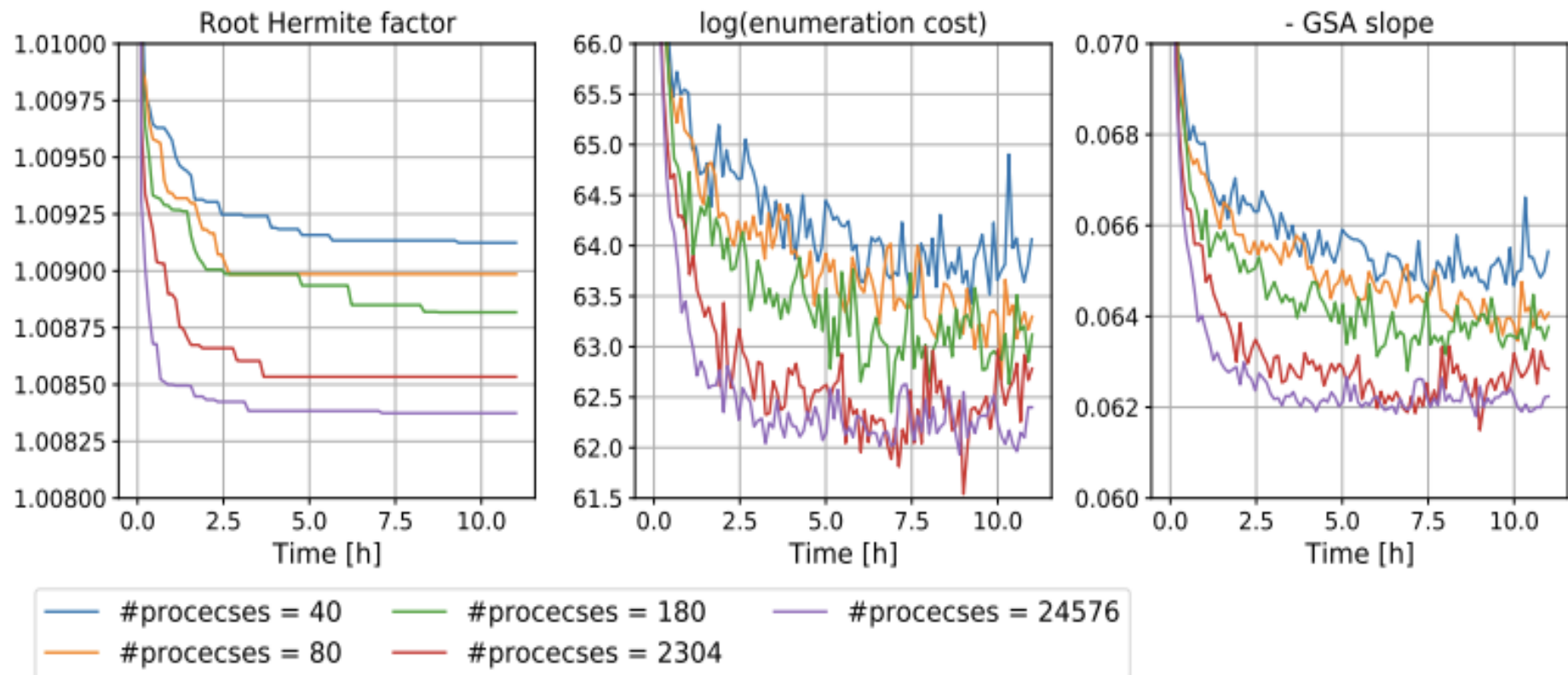
[TS+20] N. Tateiwa, Y. Shinano, S. Nakamura, A. Yoshida, S. Kaji, M. Yasuda and K. Fujisawa, Massive Parallelization for Finding Shortest Lattice Vectors Based on Ubiquity General Framework, High Performance Computing, Networking, Storage, and Analysis (SC 20).

[TS+21] N. Tateiwa, Y. Shinano, K. Yamamura, A. Yoshida, S. Kaji, M. Yasuda and K. Fujisawa, CMAP-LAP: Configurable Massively Parallel Solver for Lattice Problems, ZIB-Report 21-16 (to appear in High Performance Computing, HiPC 2021)



# Massive Parallelization of DeepBKZ (2/5)

- Efficacy of parallel sharing DeepBKZ
  - Sharing  $k = 16$  short basis vectors among solvers for dimension  $d = 120$



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

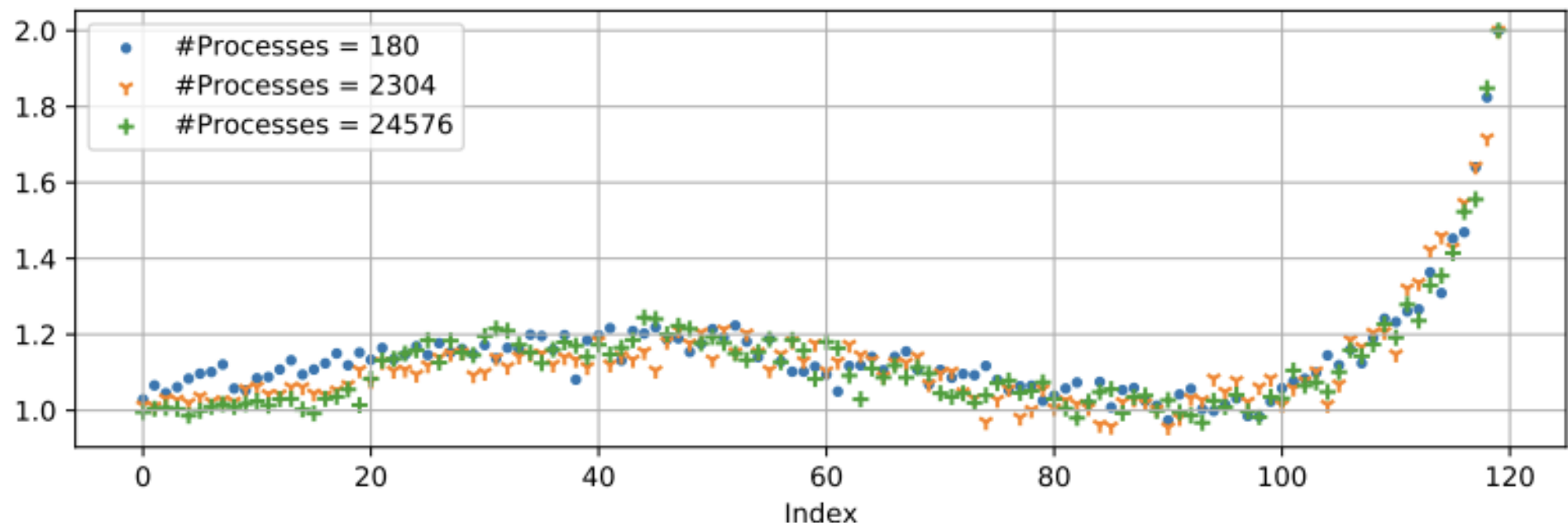
# Massive Parallelization of DeepBKZ (3/5)

- **Output quality of parallel sharing DeepBKZ**

- For an output basis  $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_n)$  of parallel sharing DeepBKZ,

- Gaps  $\frac{\|\mathbf{b}_i^*\|}{\text{GH}(\pi_i(L))}$  are shown in the below Figure

⇒ First  $k = 16$  basis vectors are close to the shortest in projected lattices





# Massive Parallelization of DeepBKZ (4/5)

- **Large-scale experiments**

- A very short lattice vector in a lattice of dimension around  $d = 130$  can be found within 100 hours on supercomputers
  - Without using the sub-sieve strategy
- **Small block sizes  $\beta = 30\text{--}40$**  are enough for parallel sharing DeepBKZ

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

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

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

<sup>†</sup> a new solution for the Darmstadt SVP challenge [26] in dimension 128 (see also Table 6 for other dimensions). \* We list the maximum number of cores and machines used for executions, including restarts, and the wall time for the updated time.

# Massive Parallelization of DeepBKZ (5/5)

- **Future Work: Use of our CMAP-LAP framework<sup>[TS+21]</sup>**
    - Supervisor-Worker parallelization type
    - **Heterogeneous execution** of lattice algorithms (Reduction, ENUM, Sieve)
    - Acceleration by asynchronously sharing lattice vectors via vector pool
- ⇒ We will embed optimal algorithms (e.g., pruned ENUM, sieve) in our framework for solving high-dimensional lattice problems

