

AH-BKZ: A Lattice Reduction Algorithm with Asynchronous Hybrid Processing

Renya Hashimoto¹[0009–0003–2492–8233], Junji Shikata²[0000–0003–2861–359X],
Atsushi Takayasu³[0000–0002–9310–6976], and
Yuntao Wang¹[0000–0002–2872–4508]

¹ The University of Electro-Communications, Tokyo, Japan

`{r-hashimoto, y-wang}@uec.ac.jp`

² Yokohama National University, Kanagawa, Japan

³ The University of Tokyo, Tokyo, Japan

Abstract. Evaluating the hardness of the Shortest Vector Problem (SVP) is essential for selecting secure parameters in lattice-based cryptography. The fastest current SVP solvers are based on variants of G6K, but their large memory consumption remains a major bottleneck, making high-dimensional executions difficult. A common strategy for alleviating this memory bottleneck is strong basis reduction preprocessing before a sieve-based algorithm. This strategy can solve SVP instances in higher dimensions than standalone G6K under the same memory constraints, but the time required for basis reduction becomes a major issue. ProPnjBKZ, a fast basis reduction algorithm optimized by Wang et al., is currently used as a promising preprocessing method for solving high-dimensional SVP instances. In this paper, we identify that, in the ProPnjBKZ-based SVP-solving pipeline, memory resources can remain underutilized during the preprocessing phase, since ProPnjBKZ requires less memory than the subsequent high-dimensional sieving process. Based on this observation, we propose *AH-BKZ*, which exploits such unused computational resources by asynchronously running multiple SVP oracles, thereby accelerating basis reduction. Our experiments show that, compared with ProPnjBKZ-based preprocessing, AH-BKZ reduces the overall runtime by up to 23.1% and by 18.6% on average among successful runs for dimensions 124–148 under comparable peak memory constraints. Moreover, in a supplementary experiment conducted in a larger computational environment, AH-BKZ set new solving records for dimension 163 in the TU Darmstadt SVP Challenge and for dimension 164 in the Ideal Lattice Challenge.

Keywords: Post-Quantum Cryptography · Lattice Algorithms · Basis Reduction · Sieve · ProPnjBKZ · G6K

1 Introduction

The standardization of post-quantum cryptography (PQC) is progressing: NIST has published ML-KEM (FIPS 203), ML-DSA (FIPS 204), and SLH-DSA (FIPS 205), with FN-DSA (FIPS 206) and HQC to follow [1]. Three of these are lattice-based. This situation highlights the importance of evaluating the concrete security of lattice-based cryptography. Secure parameter selection for lattice-based cryptography relies on concrete and reliable hardness evaluations based on practical attack algorithms. Since many hardness assumptions in lattice cryptography are reducible to variants of the approximate Shortest Vector Problem (SVP), this study focuses on practical algorithms for solving approximate SVP.

Among practical SVP solvers, Enumeration(ENUM) [14] and Sieve are two major approaches. Enumeration algorithms require only polynomial memory, but their super-exponential running time, typically $2^{O(n \log n)}$ for solving n -dimensional SVP, makes them unsuitable as standalone solvers for high-dimensional SVP instances. In contrast, sieve algorithms achieve better asymptotic running time, typically $2^{O(n)}$, and are therefore more suitable for high-dimensional SVP solving. However, this advantage comes at the cost of exponential memory consumption, also of size $2^{O(n)}$. Thus, the memory consumption of sieve algorithms becomes a major bottleneck in practical high-dimensional executions.

Since Ajtai et al. introduced sieve algorithms [2], their executable version has been proposed as Gauss Sieve in [13]. Furthermore, Progressive Sieve [12] and Dimensions for Free (d4f) [9] showed that the sieving cost in projected subspaces strongly depends on the basis quality. Therefore, strong basis reduction as preprocessing can significantly reduce the overall sieving cost.

The General Sieve Kernel (G6K) [4] is currently one of the most practical sieve frameworks and incorporates both Progressive Sieve and d4f techniques. G6K’s standard strategy, **WorkOut**, repeatedly invokes **Pump**, a sieving-based SVP-solving procedure, while gradually increasing the working dimension until the target norm is reached. Although **WorkOut** is currently one of the fastest practical SVP solvers, its enormous memory consumption becomes a critical bottleneck, making high-dimensional SVP instances difficult to solve under limited memory.

To alleviate this memory bottleneck, a common SVP-solving pipeline first performs strong basis reduction preprocessing and then applies a final high-dimensional sieve-based algorithm. Although this strategy requires substantial preprocessing time, it reduces memory consumption and enables solving higher-dimensional instances under the same memory constraints. Representative basis reduction algorithms include BKZ [14], followed by accelerated variants such as BKZ 2.0 [8], Progressive-BKZ (pBKZ) [6], and ProPnjBKZ [21]. In particular, the strategy combining ProPnjBKZ preprocessing with a sieve-based algorithm provides one of the best practical trade-offs between running time and memory usage, and is considered effective for solving high-dimensional SVP instances.

The ENUM-Sieve Reduction (ESR) algorithm [16] attempts to utilize additional computational resources by combining multiple SVP oracles, and has demonstrated promising improvements for ideal lattices. However, its scalabil-

ity to high-dimensional instances is limited because it relies on computationally expensive ENUM calls and adopts a synchronous parallel execution model.

Contributions. In this paper, we identify that, in the ProPnjBKZ-based SVP-solving pipeline, computational resources can remain underutilized during the preprocessing phase under memory constraints. In this pipeline, the memory requirement is dominated by the final high-dimensional sieve, whereas the preceding reduction phase requires substantially less memory. Consequently, when the computational environment is chosen to accommodate the peak memory usage of the final sieve, surplus resources can remain available during the reduction phase. Based on this observation, we propose Asynchronous Hybrid BKZ (*AH-BKZ*), an asynchronous basis reduction framework that combines ProPnjBKZ with multiple parallel **WorkOut** processes through systematic vector sharing. AH-BKZ is built upon the practical strategy of combining ProPnjBKZ preprocessing with high-dimensional sieve algorithms under memory constraints, and accelerates convergence by exploiting computational resources that would otherwise remain idle during the reduction phase, without increasing peak memory consumption. In other words, this work further accelerates a strong existing memory-efficient strategy through effective utilization of surplus computational resources.

Experimental results for dimensions from 124 to 148 show that, compared with the strategy of performing a high-dimensional **Pump** after ProPnjBKZ preprocessing, AH-BKZ reduces wall-clock time by up to 23.1% and by 18.6% on average while keeping peak memory consumption comparable. Furthermore, in a supplementary experiment conducted in a larger computational environment, AH-BKZ discovered shorter vectors than previously reported records for dimension 163 in the TU Darmstadt SVP Challenge and dimension 164 in the Ideal Lattice Challenge [18,17]. These record-update runs are reported as evidence of applicability rather than as a controlled comparison, since they used a different environment and execution strategy.

Positioning of this work. The contribution of this paper lies at the level of execution strategy rather than of reduction theory: AH-BKZ composes existing primitives, namely G6K [4] (**Pump** and **WorkOut**) and ProPnjBKZ [21], through an asynchronous communication layer; block reduction and the d4f technique themselves are used without modification. What is not immediate is how to convert the surplus resources of the reduction phase into reduction quality without increasing peak memory and without stalling the main reduction by synchronization; AH-BKZ shows that this conversion is achievable, and that it yields a wall-clock reduction of 18.6% on average against an already strong baseline.

2 Preliminaries

Let $\mathbf{b}_1, \dots, \mathbf{b}_n \in \mathbb{R}^m$ be linearly independent vectors. A lattice generated by these vectors is the set $L = \mathcal{L}(\mathbf{b}_1, \dots, \mathbf{b}_n) := \{\sum_{i=1}^n a_i \mathbf{b}_i \mid a_i \in \mathbb{Z}\} \subset \mathbb{R}^m$. If $n = m$, then L is called a full-rank lattice. Unless otherwise stated, we consider an n -dimensional full-rank lattice $L \subset \mathbb{R}^n$. Let $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_n)$ be a basis of L . The Gram–Schmidt orthogonalization of $\mathbf{B}$ is the orthogonal basis $\mathbf{B}^* = (\mathbf{b}_1^*, \dots, \mathbf{b}_n^*)$ defined by $\mathbf{b}_1^* = \mathbf{b}_1$ and $\mathbf{b}_i^* = \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{i,j} \mathbf{b}_j^*$ for $2 \leq i \leq n$, where $\mu_{i,j} = \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\|\mathbf{b}_j^*\|^2}$. The volume of L is defined as $\text{vol}(L) = |\det(\mathbf{B})|$. Using the Gram–Schmidt orthogonalized basis $\mathbf{B}^*$, it can also be written as $\text{vol}(L) = \prod_{i=1}^n \|\mathbf{b}_i^*\|$. For a basis $\mathbf{B}$, let $\ell_i := \log \|\mathbf{b}_i^*\|$, where $\log$ denotes the natural logarithm, and let $\bar{i} = (n+1)/2$. The slope of $\mathbf{B}$ is the least-squares slope of $(i, \ell_i)_{i=1}^n$, that is, $\text{slope}(\mathbf{B}) = (\sum_{i=1}^n i \ell_i - \bar{i} \sum_{i=1}^n \ell_i) / (\sum_{i=1}^n i^2 - n \bar{i}^2)$. For $i = 1, \dots, n$, the i -th successive minimum is defined as $\lambda_i(L) = \inf \{r > 0 \mid \dim(\text{span}\{\mathbf{v} \in L \mid \|\mathbf{v}\| \leq r\}) \geq i\}$. Let $\pi_i : \mathbb{R}^n \rightarrow \text{span}(\mathbf{b}_1, \dots, \mathbf{b}_{i-1})^\perp$ be the orthogonal projection, where $(\cdot)^\perp$ denotes the orthogonal complement. In particular, π_1 is the identity map. The projected lattice is defined as $\pi_i(L) = \{\pi_i(\mathbf{v}) \mid \mathbf{v} \in L\}$, which is an $(n-i+1)$ -dimensional lattice. Given a basis of L and an approximation factor $\gamma \geq 1$, the Approximate Shortest Vector Problem is to find a non-zero vector $\mathbf{v} \in L \setminus \{\mathbf{0}\}$ satisfying $\|\mathbf{v}\| \leq \gamma \lambda_1(L)$. The case $\gamma = 1$ is the Shortest Vector Problem (SVP). The Gaussian heuristic estimates the number of lattice points in a measurable set C by $\#(L \cap C) \approx \text{vol}(C)/\text{vol}(L)$. For a basis $\mathbf{B}$ of L , the Gaussian heuristic length is defined as $\text{GH}(\mathbf{B}) = \text{vol}(L)^{1/n}/v_n^{1/n}$, where v_n denotes the volume of the n -dimensional unit ball. Using Stirling’s formula, we obtain $\text{GH}(\mathbf{B}) \approx \sqrt{n/(2\pi e)} \text{vol}(L)^{1/n}$. For a basis $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_n)$ of an n -dimensional full-rank lattice L , the root Hermite factor is defined as $\delta(\mathbf{B}) = (\|\mathbf{b}_1\|/\text{vol}(L)^{1/n})^{1/n}$. It measures the length of the first basis vector relative to the lattice volume and is used as an indicator of reduction quality. Let $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_n)$ be a basis of L with Gram–Schmidt orthogonalization $\mathbf{B}^*$. The geometric series assumption (GSA) assumes that $\|\mathbf{b}_1^*\|, \dots, \|\mathbf{b}_n^*\|$ approximately form a geometric progression. Equivalently, there exists a constant $0 < \alpha < 1$ such that $\|\mathbf{b}_i^*\| \approx \alpha^{i-1} \|\mathbf{b}_1^*\|$ for $1 \leq i \leq n$. Under the GSA, $\log \|\mathbf{b}_i^*\|$ is approximately linear with respect to i , so that $\text{slope}(\mathbf{B}) \approx \log \alpha$. Consequently, the root Hermite factor and the slope are related by $\delta(\mathbf{B}) \approx \exp(-\text{slope}(\mathbf{B})/2)$, which is used to derive the d4f parameter in Algorithm 7. A basis $\mathbf{B}$ is size-reduced if its Gram–Schmidt coefficients satisfy $|\mu_{i,j}| \leq 1/2$ for all $1 \leq j < i \leq n$. For $\delta \in (1/4, 1]$, a basis $\mathbf{B}$ is LLL-reduced if it is size-reduced and satisfies the Lovász condition $\delta \|\mathbf{b}_{k-1}^*\|^2 \leq \|\mathbf{b}_k + \mu_{k,k-1} \mathbf{b}_{k-1}^*\|^2$ for $2 \leq k \leq n$. A basis $\mathbf{B}$ is HKZ-reduced if it is size-reduced and $\|\mathbf{b}_i^*\| = \lambda_1(\pi_i(L))$ holds for all $1 \leq i \leq n$. For $\beta \in \{2, \dots, n\}$, a basis $\mathbf{B}$ is BKZ- β -reduced if it is size-reduced and, for all $1 \leq k \leq n - \beta + 1$, the projected block $\{\pi_k(\mathbf{b}_k), \dots, \pi_k(\mathbf{b}_{k+\beta-1})\}$ is HKZ-reduced.

3 Related Works

In this section, we review state-of-the-art algorithms for solving the Shortest Vector Problem (SVP), focusing on sieve-based approaches and blockwise basis reduction algorithms.

3.1 G6K Algorithm

The General Sieve Kernel (G6K) [4], introduced by Albrecht et al. in 2019, is one of the most efficient practical frameworks for lattice sieving. A GPU-optimized implementation has also been proposed in [10]. We briefly describe its core procedures.

Pump Pump (Algorithm 1) is a core subroutine of G6K that combines the Progressive Sieve technique [12] with the Dimensions for Free (d4f) strategy [9]. Here, d4f reduces the effective sieving dimension by treating several dimensions as free, thereby lowering the cost of sieving. **Pump** performs sieving in a progressively expanding projected sublattice and evaluates candidate vectors using a scoring function. Promising vectors are inserted into appropriate positions of the basis.

Unlike global reduction algorithms, **Pump** focuses on local improvement within a projected window, enabling efficient refinement of the basis while maintaining compatibility with high-dimensional sieving.

WorkOut WorkOut (Algorithm 2) is the default strategy of G6K for solving approximate SVP instances. It repeatedly invokes **Pump** while gradually increasing the working dimension. Even if a single **Pump** execution does not immediately yield a sufficiently short vector, the basis is improved, which accelerates subsequent calls.

WorkOut is among the fastest practical SVP solvers currently known. However, since it relies on high-dimensional sieving, its exponential memory consumption remains a significant bottleneck.

3.2 BKZ Algorithm

The BKZ algorithm [14] is a blockwise basis reduction algorithm that repeatedly calls an SVP oracle within projected sublattices of dimension β , called the block size. The obtained short vector is inserted into the basis, and the process is iterated in turns.

In classical implementations, the internal SVP oracle is typically realized via enumeration (ENUM), which searches lattice points within a bounded region of the projected lattice. Increasing the block size β improves the reduction quality, but it also causes the ENUM cost to grow exponentially. BKZ 2.0 [8] mitigates this cost by introducing Extreme Pruning [11], which optimizes pruning coefficients to reduce the size of the enumeration search tree for a given

Algorithm 1 Pump [4]

Input: Basis $\mathbf{B}$, start index κ , d4f f , block size β
Output: $\mathbf{B}$

```

1:  $r \leftarrow \kappa + \beta$ 
2:  $l \leftarrow \max\{\kappa + f + 1, r - stn\}$   $\{stn \leftarrow 30: \text{start-up sieving dimension}\}$ 
3:  $ilb \leftarrow \kappa$ 
4:  $L \leftarrow \emptyset$ 
5:  $\mathbf{B}_{\pi[\kappa, r]} \leftarrow \text{LLL}(\mathbf{B}_{\pi[\kappa, r]})$ 
6: // Phase="init"
7:  $L \leftarrow \text{gauss\_sieve}(\mathbf{B}_{\pi[l, r]}, L)$ 
8: // Phase="up"
9: while  $l > \kappa + f$  do
10:    $L \leftarrow \{\text{EL}(\mathbf{v}, 1) \mid \mathbf{v} \in L\}$ 
11:    $l \leftarrow l - 1$ 
12:    $L \leftarrow \text{sieve}(\mathbf{B}_{\pi[l, r]}, L)$ 
13: end while
14: // Phase="down" ( $ds \leftarrow f$ : depth of the down phase)
15: while  $ilb < \kappa + ds$  do
16:    $BL \leftarrow \text{best\_lifts}(L)$   $\{\text{score vectors with } \text{score}(\mathbf{v}_i) \leftarrow \theta^{-i} \|\mathbf{v}_i\| / \|\mathbf{b}_i^*\|, \theta > 1$ 
     $\text{fixed}\}$ 
17:   if  $BL \neq \emptyset$  then
18:      $ii \leftarrow \text{index}(\max(BL))$   $\{\text{Find the best scoring position}\}$ 
19:     Insert  $\mathbf{v}_{ii}$  into  $\mathbf{B}_{\pi[\kappa, r]}$ 
20:      $ilb \leftarrow ii + 1$ 
21:   else
22:      $L \leftarrow \{\text{SL}(\mathbf{v}, 1) \mid \mathbf{v} \in L\}$ 
23:   end if
24:    $L \leftarrow \text{sieve}(\mathbf{B}_{\pi[l, r]}, L)$ 
25:    $l \leftarrow l + 1$ 
26: end while
27: return  $\mathbf{B}$ 

```

success probability. Several further improvements of BKZ have been proposed in [23,3,22].

Another important aspect of BKZ-based reduction is the choice of the block-size schedule. Progressive-BKZ (pBKZ) [6], proposed by Aono et al., uses a progressive strategy in which the block size is gradually increased during the reduction process, rather than being fixed throughout all tours. The block-size schedule is selected using a simulator to approximately minimize the estimated computational cost by exploiting the dependence of the ENUM cost on the current basis quality.

Algorithm 2 WorkOut [4]

Input: Basis $\mathbf{B}$, start index κ , block size β , d4f step f^+ , minimum d4f $f_{\min}$
Output: $\mathbf{B}$

```

1:  $f \leftarrow \beta - f^+$ 
2: while  $f > f_{\min}$  do
3:    $\mathbf{B} \leftarrow \text{Pump}(\mathbf{B}, \kappa, f, \beta)$ 
4:   if a target short vector has been obtained then
5:     break
6:   end if
7:    $f \leftarrow f - f^+$ 
8: end while
9: return  $\mathbf{B}$ 

```

3.3 ProPnjBKZ Algorithm

ProPnjBKZ [21] is based on the Pump-based lattice reduction framework introduced in G6K by Albrecht et al. [4]. Xia et al. proposed LWE-solving and SVP-solving strategies that combine progressive PnjBKZ preprocessing with a final high-dimensional Pump.

Algorithm 3 describes one PnjBKZ tour with block size β . PnjBKZ is a BKZ-type lattice reduction algorithm that uses Pump as the SVP oracle for local blocks. For sufficiently large block sizes, Pump enables faster reduction than enumeration, allowing larger block sizes within practical running times. Furthermore, PnjBKZ introduces a jump strategy. While standard BKZ-type algorithms shift the starting index of local blocks by one, the jump strategy shifts it by j . This reduces the number of Pump calls in each tour. Although this heuristic may slightly degrade the reduction quality of a single tour, it significantly accelerates the overall reduction process. Later, Wang et al. developed a PnjBKZ simulator that considers the jump strategy [20]. This simulator estimates the block size required to achieve reduction quality comparable to that of a standard pBKZ tour under a given jump parameter.

The strategy of performing a high-dimensional Pump after ProPnjBKZ preprocessing is regarded as one of the most practical time-memory trade-offs for solving SVP. In fact, it has been reported to update records up to dimension 170 in the SVP Challenge and up to dimension 176 in the Ideal Lattice Challenge [20]. AH-BKZ differs from distributed sieving [5] and from ESR [16] in how multiple processes are coordinated; we detail the comparison in Appendix A.

4 Our Proposal

In this section, we describe the design objective, the internal vector-sharing mechanism, and the overall SVP-solving pipeline of *AH-BKZ*.

Algorithm 3 PumpNJumpTour [4]

Input: Basis $\mathbf{B}$, block size β' , d4f f' , jump size j **Output:** $\mathbf{B}$

```

1:  $\kappa \leftarrow 0, \beta \leftarrow \beta' - f', f \leftarrow 0$ 
2: // Gradually increase the block size
3: while  $f < f'$  do
4:    $\mathbf{B} \leftarrow \text{Pump}(\mathbf{B}, \kappa, f, \beta)$ 
5:    $\Delta \leftarrow \min(j, f' - f)$ 
6:    $f \leftarrow f + \Delta, \beta \leftarrow \beta + \Delta$ 
7: end while
8: // Slide the window
9: while  $\kappa < n - 1$  do
10:   $\mathbf{B} \leftarrow \text{Pump}(\mathbf{B}, \kappa, f', \min(\beta', n - \kappa))$ 
11:   $\kappa \leftarrow \kappa + j$ 
12: end while
13: return  $\mathbf{B}$ 

```

4.1 Design Objective and Overview of AH-BKZ

In the SVP-solving pipeline considered in this work, ProPnjBKZ preprocessing is followed by a final high-dimensional Pump. The most memory-intensive sieving phase occurs in the final Pump stage. Since the space complexity of lattice sieving grows exponentially with the lattice dimension, executing the final high-dimensional Pump requires a computational environment with enough memory. In contrast, during the preceding ProPnjBKZ reduction stage, the available computational resources are not always fully utilized. In a standard ProPnjBKZ-based pipeline, GPU-side basis reduction is the main process, while CPU resources can remain underutilized during preprocessing. The design question we address is therefore how to convert these surplus resources into reduction quality subject to two constraints: the additional processes must together stay below the memory footprint of the final high-dimensional Pump, and they must not be coupled to the main reduction by global synchronization, which would make every exchange wait for the slowest process. We therefore extend ProPnjBKZ with asynchronous hybrid processing, in which additional SVP oracles run concurrently with the basis reduction and exchange short basis vectors with it in a non-blocking manner. This extended framework is AH-BKZ. AH-BKZ executes ProPnjBKZ as the main reduction process, while additional SVP-based processes run asynchronously in the background. Each process searches for short vectors and shares the obtained vectors so that they can be used to improve the bases maintained by other processes. The goal of AH-BKZ is to accelerate basis-quality improvement during preprocessing by exploiting otherwise idle computational resources, without exceeding the memory budget required by the final high-dimensional Pump.

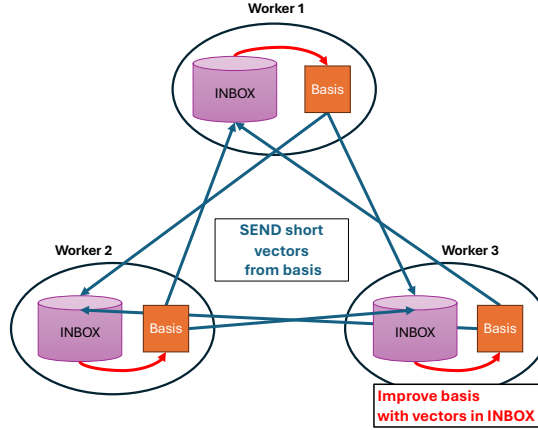


Fig. 1: Conceptual image of vector sharing between Workers.

4.2 Workers and Vector Sharing

AH-BKZ avoids global synchronization by introducing independent *Workers*. Each Worker maintains its own lattice basis and a buffer of received vectors, called an **INBOX**. Workers execute SVP or reduction subroutines asynchronously and periodically exchange short vectors, as illustrated in Figure 1.

Each Worker performs three operations: **SEND**, **RETRIEVE**, and **MERGE**. In **SEND**, a Worker sends short vectors in its current basis to the **INBOXes** of other Workers; in our implementation, each Worker sends the five shortest vectors in its current basis. In **RETRIEVE**, a Worker retrieves the vectors accumulated in its own **INBOX**. In **MERGE**, the retrieved vectors are integrated into its own basis, and the reduction quality is restored as shown in Algorithm 4. The block size β used for this restoration is a parameter of **MERGE**, and we set $\beta = 30$ for all Workers in our implementation.

Since Workers operate asynchronously, they do not need to stop at the same time to communicate or merge vectors. When a Worker discovers a short vector, the vector is sent to the **INBOXes** of other Workers, and each receiving Worker retrieves and merges it at its own timing. Therefore, AH-BKZ can exploit discovered short vectors without waiting for slower Workers to finish. Moreover, each Worker maintains an independent basis and updates it at different timings, enabling continuous basis improvement during the reduction process.

4.3 AH-BKZ Preprocessing

AH-BKZ preprocessing runs the **ProPnjWorker** and multiple **CPUG6KWorkers** asynchronously, sharing short vectors to improve basis quality. The **ProPnjWorker** is the main reduction process in AH-BKZ and runs **ProPnjBKZ** on a GPU. Its basic behavior follows **ProPnjBKZ**, but vector-sharing operations are added at

Algorithm 4 MERGE

Input: Basis $\mathbf{B}$, received vectors W , block size β for re-reduction
Output: $\mathbf{B}$

- 1: $X \leftarrow \emptyset$
- 2: Sort W in ascending order of norm
- 3: **for all** $\mathbf{w} \in W$ **do**
- 4: Find the smallest index i such that $\|\mathbf{w}\| < \|\mathbf{b}_i\|$
- 5: **if** such i exists **then**
- 6: Insert $\mathbf{w}$ into $\mathbf{B}$ at position i
- 7: **else**
- 8: $X \leftarrow X \cup \{\mathbf{w}\}$
- 9: **end if**
- 10: **end for**
- 11: Sort X in ascending order of norm
- 12: Append the vectors in X to $\mathbf{B}$
- 13: // Restore linear independence and re-reduce the basis
- 14: $\mathbf{B} \leftarrow \text{BKZ2.0}(\mathbf{B}, \beta, \delta = 0.99)$
- 15: **return** $\mathbf{B}$

the end of each BKZ tour. After each tour, the **ProPnjWorker** selects short vectors from its current basis and sends them to the **INBOXes** of other Workers. It then retrieves the vectors accumulated in its own **INBOX**, integrates them into its basis, restores the reduction quality of the merged basis, and proceeds to the next tour. Thus, the **ProPnjWorker** performs the usual progressive reduction of ProPnjBKZ while asynchronously incorporating short vectors discovered by other Workers. The progressive block-size strategy used by the **ProPnjWorker** is adjusted to account for improvements obtained through asynchronous vector sharing; the details are given in Appendix B.

A **CPUG6KWorker** is an auxiliary short-vector search process executed on a CPU. Multiple **CPUG6KWorkers** repeatedly execute **WorkOut** with gradually increasing subspace dimensions. We use **WorkOut** because it can efficiently find short vectors and directly benefits from basis improvements received from the **ProPnjWorker**. After completing a processing unit, each **CPUG6KWorker** sends short vectors in its current basis to other Workers, retrieves the vectors accumulated in its own **INBOX**, and merges them into its basis. If the merged vectors improve the basis quality, subsequent **WorkOut** executions may search for short vectors more effectively. This process is executed without synchronization with the **ProPnjWorker**; hence, a **CPUG6KWorker** can continue short-vector search without waiting for GPU-side ProPnjBKZ tours to finish, and the **ProPnjWorker** does not need to wait for CPU-side **WorkOut** executions.

In this way, AH-BKZ preprocessing combines the strong reduction capability of GPU-based ProPnjBKZ with the auxiliary short-vector search capability of CPU-based **WorkOut**. The **ProPnjWorker** can accelerate basis improvement by incorporating short vectors discovered by CPU Workers. Conversely,

Algorithm 5 ProPnjWorker

Input: Basis $\mathbf{B}$, initial block size β_p , maximum block size $\beta_{p_{\max}}$
Output: $\mathbf{B}$

- 1: $\mathbf{B} \leftarrow \text{BKZ2.0}(\mathbf{B}, \beta = 30, \delta = 0.99)$
- 2: **while** $\beta_p \leq \beta_{p_{\max}}$ **do**
- 3: $\mathbf{B} \leftarrow \text{PumpNJumpTour}(\mathbf{B}, \beta' = \beta_p, f' = 1, j = 9)$
- 4: **SEND** short vectors to the **INBOXes** of the other Workers
- 5: **RETRIEVE** new vectors V_{new} from its own **INBOX**
- 6: $\mathbf{B} \leftarrow \text{MERGE}(\mathbf{B}, V_{\text{new}}, \beta = 30)$
- 7: Increase β_p following the progressive strategy
- 8: **end while**
- 9: Signal **STOP** to all CPUG6KWorkers
- 10: **return** $\mathbf{B}$

Algorithm 6 CPUG6KWorker

Input: Basis $\mathbf{B}$, initial dimension β_c , maximum dimension $\beta_{c_{\max}}$
Output: $\mathbf{B}$

- 1: **while** $\beta_c \leq \beta_{c_{\max}}$ **and not** **STOP_SIGNAL** **do**
- 2: $\mathbf{B} \leftarrow \text{BKZ2.0}(\mathbf{B}, \beta = 30, \delta = 0.99)$
- 3: $\mathbf{B} \leftarrow \text{WorkOut}(\mathbf{B}, \kappa = 0, \beta = \beta_c, f^+ = 2, f_{\min} = 0)$
- 4: **SEND** short vectors to the **INBOXes** of the other Workers
- 5: **RETRIEVE** new vectors V_{new} from its own **INBOX**
- 6: $\mathbf{B} \leftarrow \text{MERGE}(\mathbf{B}, V_{\text{new}}, \beta = 30)$
- 7: Increase β_c by 10
- 8: **end while**
- 9: **return** $\mathbf{B}$

CPUG6KWorkers can also benefit from improved short vectors shared by the ProPnjWorker, which may make subsequent searches more effective. In our experiments, the maximum sieving dimension actually used by the CPUG6KWorkers is at least 10 dimensions lower than that used by the final high-dimensional Pump. Consequently, the additional memory consumption introduced by AH-BKZ preprocessing does not increase the overall peak memory usage.

4.4 Overall SVP-Solving Pipeline

The proposed SVP-solving pipeline consists of AH-BKZ preprocessing followed by a final high-dimensional Pump. First, AH-BKZ preprocessing executes asynchronous hybrid basis reduction as an extension of ProPnjBKZ and improves the quality of the input basis. Then, a high-dimensional Pump is executed on the resulting basis to search for the target short vector. Algorithm 7 describes the SVP-solving procedure using AH-BKZ, while Algorithms 5 and 6 provide the pseudocode for each Worker.

Algorithm 7 AH-BKZ + Pump

Input: Basis $\mathbf{B}$, approximation factor γ , number of CPU Workers N_{cpu}
Output: $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ with $\|\mathbf{v}\| \leq \gamma \cdot \text{GH}(\mathbf{B})$, or $\perp$

- 1: $\mathbf{B} \leftarrow \text{LLL}(\mathbf{B})$
- 2: // Launch AH-BKZ workers
- 3: Run $\text{ProPnjWorker}(\mathbf{B})$ asynchronously
- 4: **for** $i \leftarrow 1$ **to** N_{cpu} **do**
- 5: Generate a random unimodular matrix $\mathbf{U}_i$
- 6: $\mathbf{B}_i \leftarrow \mathbf{B}\mathbf{U}_i$
- 7: Run $\text{CPUG6KWorker}(\mathbf{B}_i)$ asynchronously
- 8: **end for**
- 9: // Wait for completion of ProPnjBKZ
- 10: Wait until ProPnjWorker finishes
- 11: Terminate all CPUG6KWorker instances
- 12: $\mathbf{B} \leftarrow \text{ProPnjWorker}.\mathbf{B}$
- 13: // Final high-dimensional Pump
- 14: $\tau \leftarrow \gamma \cdot \text{GH}(\mathbf{B})$ {target norm}
- 15: $s \leftarrow \text{slope}(\mathbf{B})$
- 16: $\delta \leftarrow \exp(-s/2)$ {root Hermite factor under the GSA (Section 2)}
- 17: // Wang et al.'s optimistic d4f estimate [20]
- 18: $f \leftarrow \left\lfloor \log_{\delta} \sqrt{4/3} \right\rfloor$
- 19: $\mathbf{B} \leftarrow \text{Pump}(\mathbf{B}, 0, f, n)$
- 20: $\mathbf{v} \leftarrow \text{min}(\mathbf{B})$
- 21: **if** $\|\mathbf{v}\| > \tau$ **then**
- 22: **return** $\perp$ {the optimistic d4f was too aggressive; the run is counted as a failure}
- 23: **end if**
- 24: **return** $\mathbf{v}$

5 Experiments

In this section, we evaluate *AH-BKZ* from two viewpoints. First, we compare its running time and memory usage with G6K and ProPnjBKZ on SVP Challenge instances from dimension 124 to 148. Second, as a supplementary experiment, we report record-update runs on higher-dimensional SVP and Ideal Lattice Challenge instances.

5.1 Experimental Setup

We evaluated AH-BKZ on SVP Challenge instances from dimension 124 to 148. We focus on this dimension range because the main objective of this study is to improve the running time of high-dimensional SVP solving while keeping memory usage comparable to the ProPnjBKZ-based pipeline. For each dimension, we used 10 independent seeds and compared three methods: G6K, ProPnjBKZ,

and AH-BKZ. Here, G6K denotes the `WorkOut` strategy in G6K. ProPnjBKZ denotes ProPnjBKZ preprocessing followed by a final high-dimensional `Pump`. AH-BKZ denotes the proposed method with two CPU workers, followed by a final high-dimensional `Pump`.

Although the advantage of ProPnjBKZ over standalone G6K in terms of the time-memory trade-off has already been shown by Wang et al. [20], we include G6K results for reference. This comparison helps position AH-BKZ among practical SVP solvers. Since `ProPnjWorker` behaves identically to ProPnjBKZ when run without CPU workers, this setting is used as the ProPnjBKZ baseline.

Our implementation is based on `pro-pnj-bkz` [19]. The `CPUG6KWorker` uses the CPU version of G6K. BKZ 2.0 from `fpv111` [15] is used for preprocessing and for the small block-size BKZ operations in `MERGE`. We conducted the experiments on a machine equipped with an Intel Xeon Platinum 8360Y CPU, two NVIDIA A100 GPUs, and 112 GiB of RAM. All methods were executed on the same machine and under the same memory budget. For each run, we measured the wall-clock time T_W , CPU time T_{CPU} , GPU time T_{GPU} , and peak memory usage. The running times were originally measured in seconds and are reported in hours in Table 1. The memory usage is reported in GiB. A run was considered successful if the output vector $\mathbf{v}$ satisfied $\|\mathbf{v}\| \leq 1.05 \cdot \text{GH}(\mathbf{B})$. In the performance table, we report the average T_W , T_{CPU} , T_{GPU} , and peak memory usage over successful runs only.

5.2 Performance Comparison

Table 1 reports conditional averages over successful runs only. Therefore, the values should be interpreted together with the success rates reported in Section 5.3. Here, T_W denotes the wall-clock time, T_{CPU} denotes the CPU time, T_{GPU} denotes the GPU time, and S denotes the peak memory usage.

Figure 2 shows the relationship between the average wall-clock time and the average peak memory usage over successful runs. The horizontal axis represents wall-clock time, and the vertical axis represents peak memory usage. The color indicates the lattice dimension, and the marker indicates the method.

The figure shows that G6K achieves short wall-clock time in lower dimensions, but its peak memory usage increases rapidly as the dimension grows. In contrast, ProPnjBKZ and AH-BKZ maintain relatively low peak memory usage even in higher dimensions, because their stronger preprocessing reduces the memory requirement of the final high-dimensional `Pump`. Among these memory-efficient approaches, AH-BKZ is located on the shorter wall-clock-time side compared with ProPnjBKZ. This tendency shows that AH-BKZ improves the running time of the ProPnjBKZ-based SVP-solving pipeline while preserving its memory-efficient behavior.

The quantitative improvements are evaluated using the conditional averages reported in Table 1. Among successful runs, AH-BKZ achieves a maximum wall-clock time reduction of 23.1% at dimension 124 and an average reduction of approximately 18.6% compared with ProPnjBKZ. For CPU time, the maximum

Table 1: Average performance over successful runs.

Dim	Method	T_W/h	T_{CPU}/h	T_{GPU}/h	S/GiB
124	G6K	0.0836	0.5841	0.0816	2.717
	ProPnjBKZ	0.2639	1.3555	0.2628	2.696
	AH-BKZ	0.2030	1.0672	0.1961	2.693
128	G6K	0.1295	1.1700	0.1272	4.201
	ProPnjBKZ	0.4115	2.4154	0.4104	4.076
	AH-BKZ	0.3441	2.0885	0.3361	4.070
132	G6K	0.1834	1.8913	0.1809	5.809
	ProPnjBKZ	0.5683	3.5265	0.5669	5.159
	AH-BKZ	0.4853	3.1038	0.4766	5.191
136	G6K	0.3183	3.8322	0.3155	9.316
	ProPnjBKZ	0.8760	5.7245	0.8744	6.577
	AH-BKZ	0.7311	4.9127	0.7211	6.582
140	G6K	0.5574	7.3630	0.5542	16.231
	ProPnjBKZ	1.0831	7.1786	1.0812	7.474
	AH-BKZ	0.8727	5.9464	0.8611	7.450
144	G6K	1.0857	14.8539	1.0820	71.997
	ProPnjBKZ	2.6827	18.7940	2.6805	34.745
	AH-BKZ	2.0822	14.7376	2.0663	34.649
148	G6K	1.7187	24.1176	1.7145	92.645
	ProPnjBKZ	4.7360	34.3371	4.7333	60.542
	AH-BKZ	3.8999	28.5271	3.8823	60.416

reduction reaches 21.6% at dimension 144, with an average reduction of approximately 16.7%. For GPU time, the maximum reduction reaches 25.4% at dimension 124, with an average reduction of approximately 19.7%. Since the computational environment is provisioned to accommodate the peak memory usage of the final high-dimensional Pump, the CPU resources used by the auxiliary Workers are already allocated for the entire duration of the run, whether or not they are used. The relevant cost measure under this provisioning is the wall-clock time of the allocated machine rather than the accumulated CPU time, so the wall-clock reduction translates directly into a reduction of the total resource cost. In our experiments T_{CPU} also decreased in every tested dimension, from 12.0% at dimension 132 to 21.6% at dimension 144. Regarding memory, the relative difference in peak memory usage between AH-BKZ and ProPnjBKZ lies within $[-0.32\%, +0.62\%]$ over all tested dimensions; the largest increase is $+0.62\%$ at dimension 132 (5.191 vs. 5.159 GiB), and AH-BKZ uses strictly less memory than ProPnjBKZ in five of the seven dimensions. Since this difference is far below the run-to-run fluctuation of the peak memory measurement, we conclude that AH-BKZ attains the above speedups without increasing the peak memory usage of the ProPnjBKZ-based pipeline.

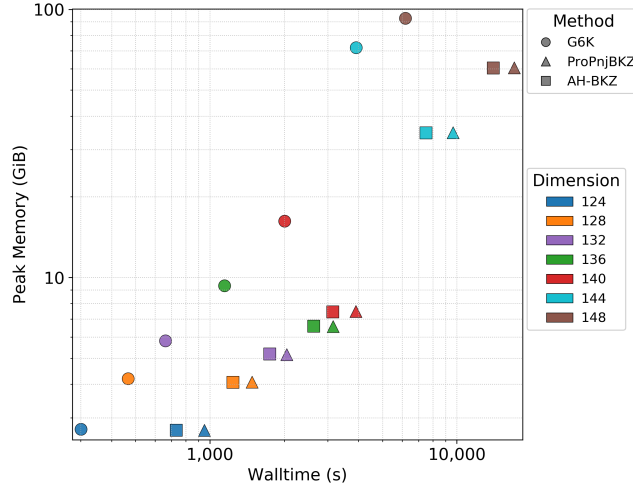


Fig. 2: Comparison of average wall-clock time and peak memory usage over successful runs.

Table 2: Number of successful runs out of 10 trials for each dimension.

Method	124	128	132	136	140	144	148
G6K	10	10	10	10	10	6	1
ProPnjBKZ	6	7	5	5	4	5	4
AH-BKZ	3	6	5	6	3	4	2

5.3 Success Rate

In this experiment, the dimension-for-free parameter was set relatively optimistically. This reduces the effective sieving dimension, but it can also make the final **Pump** fail to output a vector satisfying the target norm. As a result, some runs did not output a vector satisfying the target norm in the final **Pump** phase. Therefore, the average wall-clock time, CPU time, GPU time, and peak memory usage were computed only over successful runs.

We evaluated the success rate using 10 independent seeds for each dimension. Table 2 shows the number of successful runs for each method and dimension. Here, a successful run means that the output vector $\mathbf{v}$ satisfies $\|\mathbf{v}\| \leq 1.05 \cdot \text{GH}(\mathbf{B})$.

The main purpose of this experiment is not to maximize the success rate itself, but to compare ProPnjBKZ and AH-BKZ under the same conditions and evaluate the running time and memory behavior among successful runs. Moreover, since each setting was evaluated with only 10 independent trials, the relative success rates should not be regarded as a reliable performance metric, considering the randomness of sieve-based algorithms. Therefore, we report the success

rate as supplementary information for interpreting the conditional performance comparison.

A success-probability-aware comparison between ProPnjBKZ and AH-BKZ in terms of the expected time-to-success is given in Appendix C, where we also explain why G6K is excluded from that comparison. The differences observed there are attributable to the success counts of Table 2, which the present number of trials does not resolve.

In G6K, the failures in higher dimensions were mainly caused by memory consumption exceeding the available memory capacity.

5.4 Record Updates on the SVP Challenge Series

As a supplementary experiment, we evaluated AH-BKZ in record-update runs under a larger computational environment. The purpose of this experiment is not to compare AH-BKZ with ProPnjBKZ, but to confirm that the proposed framework remains applicable to instances substantially higher-dimensional than those in Sections 5.1–5.3. Using the proposed AH-BKZ method, we updated the solving records for multiple instances up to dimension 163 in the SVP Challenge. Furthermore, we achieved record updates for multiple instances up to dimension 164 in the Ideal Lattice Challenge [18,17]. We conducted the record-update experiments on a system equipped with two AMD EPYC 7643 CPUs and six NVIDIA A100 GPUs, with a total RAM capacity of 953 GiB. These experiments used a different computational environment and a modified execution strategy from the controlled comparison experiments in Sections 5.1–5.3.

Consequently, these results do not constitute a controlled comparison against ProPnjBKZ, since no baseline was executed under the same environment and execution strategy; none of the quantitative performance claims of this paper depends on them. For the record-setting runs, we adopted a modified execution strategy. After performing basis reduction with AH-BKZ, we iteratively executed **Pump** while gradually decreasing the **d4f** parameter until the target norm was achieved. This strategy is similar in spirit to **WorkOut** (Algorithm 2) and is intended to increase the success probability in higher dimensions. In particular, for the 163-dimensional SVP Challenge instance, we obtained a vector of norm 3333.1 with a GH ratio of 1.037. According to the run summary, the wall-clock time was 14.84 hours, the CPU time was 227.47 CPU hours, the GPU time was 14.81 GPU hours, and the peak memory usage was 594.9 GiB. For the 164-dimensional Ideal Lattice Challenge instance, we obtained a vector of norm 3315.3 with a GH ratio of 1.027. According to the run summary, the wall-clock time was 22.17 hours, the CPU time was 349.04 CPU hours, the GPU time was 22.14 GPU hours, and the peak memory usage was 916.7 GiB. These record updates indicate that AH-BKZ is applicable in practice to SVP instances well beyond the dimensions used in the controlled comparison, provided that sufficiently large computational resources are available.

Acknowledgement. This work was partially supported by JSPS KAKENHI Grant Number JP25K21805, JP26K02909, JP26K02910, JST K Program Grant Number JPMJKP24U2, Japan.

References

1. US Department of Commerce, National Institute of Standards and Technology. Post-Quantum Cryptography, 2026. <https://csrc.nist.gov/projects/post-quantum-cryptography/>.
2. M. Ajtai, R. Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing*, pages 601–610, 2001.
3. M. R. Albrecht, S. Bai, J. Li, and J. Rowell. Lattice reduction with approximate enumeration oracles - practical algorithms and concrete performance. In T. Malkin and C. Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Proceedings, Part II*, volume 12826 of *Lecture Notes in Computer Science*, pages 732–759. Springer, 2021.
4. M. R. 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 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part II*, pages 717–746, 2019.
5. M. R. Albrecht and J. Rowell. Scaling lattice sieves across multiple machines. *IACR Communications in Cryptology*, 1(4), 2025.
6. Y. Aono, Y. Wang, T. Hayashi, and T. Takagi. Improved progressive BKZ algorithms and their precise cost estimation by sharp simulator. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part I*, pages 789–819, 2016.
7. Y. Aono, Y. Wang, T. Hayashi, and T. Takagi. The progressive BKZ code. Available at <http://www2.nict.go.jp/security/pbkzcode/>, 2017.
8. Y. Chen and P. Q. Nguyen. BKZ 2.0: Better lattice security estimates. In D. H. Lee and X. Wang, editors, *Advances in Cryptology - ASIACRYPT 2011*, pages 1–20. Springer Berlin Heidelberg, 2011.
9. L. Ducas. Shortest vector from lattice sieving: A few dimensions for free. In J. B. Nielsen and V. Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part I*, volume 10820 of *Lecture Notes in Computer Science*, pages 125–145. Springer, 2018.
10. L. Ducas, M. Stevens, and W. P. J. van Woerden. Advanced lattice sieving on GPUs, with tensor cores. In A. Canteaut and F. Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021*, volume 12697 of *Lecture Notes in Computer Science*, pages 249–279. Springer, 2021.
11. N. Gama, P. Q. Nguyen, and O. Regev. Lattice enumeration using extreme pruning. In *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings*, pages 257–278, 2010.
12. T. Laarhoven and A. Mariano. Progressive lattice sieving. In T. Lange and R. Steinwandt, editors, *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018, Proceedings*, volume 10786 of *Lecture Notes in Computer Science*, pages 292–311. Springer, 2018.
13. D. Micciancio and P. Voulgaris. Faster exponential time algorithms for the shortest vector problem. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010*, pages 1468–1480, 2010.

14. C. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Math. Program.*, 66:181–199, 1994.
15. The FPYLLL development team. fpylll, a Python (2 and 3) wrapper for fplll. Available at <https://github.com/fplll/fpylll>, 2017.
16. K. Toda and Y. Wang. Proposal of an SVP solver on prime cyclotomic lattices. In G. Yang, S. Liu, C. Su, A. Otsuka, and Z. Lian, editors, *Provable and Practical Security - 19th International Conference, ProvSec 2025, Proceedings*, volume 16172 of *Lecture Notes in Computer Science*, pages 166–182. Springer, 2025.
17. TU Darmstadt. Ideal Lattice Challenge. Available at <https://www.latticechallenge.org/ideallattice-challenge>, 2026.
18. TU Darmstadt. SVP Challenge. Available at <https://www.latticechallenge.org/svp-challenge>, 2026.
19. L. Wang. pro-pnj-bkz: An improved progressive BKZ implementation with pump and jump. Software accompanying the work of Wang et al. Available at <https://github.com/Summwer/pro-pnj-bkz>, 2023.
20. L. Wang, Y. Wang, and B. Wang. A trade-off SVP-solving strategy based on a sharper pnj-BKZ simulator. In J. K. Liu, Y. Xiang, S. Nepal, and G. Tsudik, editors, *Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security, ASIA CCS 2023, Melbourne, VIC, Australia, July 10-14, 2023*, pages 664–677. ACM, 2023.
21. W. Xia, L. Wang, G. Wang, D. Gu, and B. Wang. A refined hardness estimation of LWE in Two-Step mode. In Q. Tang and V. Teague, editors, *Public-Key Cryptography – PKC 2024*, pages 3–35, Cham, 2024. Springer Nature Switzerland.
22. K. Yamamura, Y. Wang, and E. Fujisaki. Improved lattice enumeration algorithms by primal and dual reordering methods. *IET Information Security*, 17:35–45, 2022.
23. M. Yasuda, S. Nakamura, and J. Yamaguchi. Analysis of DeepBKZ reduction for finding short lattice vectors. *Des. Codes Cryptogr.*, 88(10):2077–2100, 2020.

A Distinguishing AH-BKZ from Existing Resource-Aware Approaches

A related direction is to combine multiple SVP oracles to utilize heterogeneous computational resources. Toda and Wang [16] proposed ESR, where the basis is split and different SVP oracles are executed in parallel. However, its synchronous execution model makes each cycle depend on the slowest oracle, whereas AH-BKZ lets each Worker proceed independently without a global synchronization point. Moreover, ESR executes oracles on split bases, while AH-BKZ preserves the ProPnjBKZ reduction flow on the main basis through the `ProPnjWorker`. Finally, ESR relies on ENUM as one of its internal oracles, whose cost grows rapidly with the sublattice dimension, whereas AH-BKZ uses auxiliary `WorkOut` processes to search for short vectors asynchronously.

Another related direction is distributed lattice sieving. Distributed lattice sieving approaches, such as the work of Albrecht and Rowell [5], aim to scale a single sieving computation by distributing large sieving databases across multiple machines. However, sieve algorithms require moving and managing a large number of lattice vectors, and their scalability is limited by communication overhead. In contrast, AH-BKZ does not aim to improve or scale a single sieving computation itself. Rather, it improves the execution strategy of the overall SVP-solving

Table 3: ProPnjBKZ block-size strategy with jump size 9.

Dim	Block size
120	63 → 66 → 68 → 79 → 87 → 97 → 108 → 114
130	66 → 66 → 66 → 66 → 73 → 78 → 87 → 93 → 103 → 112 → 123
140	66 → 68 → 69 → 76 → 79 → 87 → 92 → 100 → 110 → 119 → 128
150	66 → 68 → 69 → 76 → 79 → 87 → 88 → 93 → 99 → 109 → 118 → 127 → 138
160	66 → 72 → 72 → 78 → 87 → 89 → 92 → 96 → 106 → 114 → 125 → 135 → 144

The strategy starts from a BKZ-30-reduced basis. Each number denotes one BKZ tour with the indicated block size.

pipeline by exploiting computational resources that are underutilized during ProPnjBKZ reduction. Specifically, AH-BKZ shares only short vectors among processes instead of sharing an entire large sieving database, thereby reducing communication and data-movement overhead while still propagating useful information for basis improvement. We do not provide a direct experimental comparison with either approach.

The ESR implementation of [16] is not publicly available, and a re-implementation would require us to fix its pruning parameters and internal oracles ourselves, which would make the measurement a comparison of implementations rather than of designs. The distributed sieving of [5] is evaluated on a multi-machine cluster, whereas all our experiments run on a single node, and running it on one node would remove exactly the property that the approach is designed to exploit. We therefore restrict the comparison to the design level.

B Progressive Strategy Adjustment

The progressive strategy used in the ProPnjWorker is generated using Aono et al.’s library [7] and Wang et al.’s strategy generator [20]. An example of the simulated progressive block-size strategy with jump size 9 is shown in Table 3. Each entry in the sequence corresponds to one BKZ tour with the indicated block size.

To account for basis improvements resulting from shared vectors, we partially skip tours in the progressive schedule. Let $S = [\beta_1, \beta_2, \dots, \beta_N]$ denote the sequence of BKZ tours, where β_i is the block size used in the i -th tour. Excluding the first tour, we skip alternating tours as follows:

$$\begin{aligned} \text{If } N \text{ is even: } & [\beta_1, \beta_2, \beta_3, \dots, \beta_N] \rightarrow [\beta_1, \beta_2, \beta_4, \dots, \beta_N], \\ \text{If } N \text{ is odd: } & [\beta_1, \beta_2, \beta_3, \dots, \beta_N] \rightarrow [\beta_1, \beta_3, \beta_5, \dots, \beta_N]. \end{aligned}$$

This adjustment reduces redundant tours while preserving reduction strength, since intermediate improvements are injected asynchronously by other Workers.

Table 4: Expected time-to-success $\mathbb{E}[T]$ in hours, computed over all 10 trials for each dimension.

Method	124	128	132	136	140	144	148
ProPnjBKZ	0.440	0.589	1.138	1.751	2.710	5.372	11.935
AH-BKZ	0.670	0.573	0.968	1.217	2.914	5.195	19.808

C Expected Time-to-Success

The averages in Table 1 are conditional on success, so they do not account for the runs that failed to reach the target norm. As a success-probability-aware measure, we report the expected time-to-success

$$\mathbb{E}[T] = \frac{\sum_i T_{W,i}}{\#\{\text{successful runs}\}},$$

where the numerator sums the wall-clock time $T_{W,i}$ over all 10 trials of the corresponding setting, including the failed ones, and the denominator is the number of those 10 trials that reached the target norm. Under the provisioning discussed in Section 5.2, where the machine is chosen to accommodate the peak memory usage of the final high-dimensional **Pump**, this quantity is the expected wall-clock time for which the allocated machine is occupied per solved instance.

We restrict this comparison to ProPnjBKZ and AH-BKZ. The failures of these two methods share a common cause: the run completes the final high-dimensional **Pump** and returns $\perp$ because the optimistically chosen `d4f` parameter was too aggressive, so the measured wall-clock time of a failed run is the time of a completed execution. The failures of G6K in higher dimensions instead arise from memory consumption exceeding the available capacity, and the corresponding wall-clock time is the time until the run was aborted rather than the time of a completed execution. Including G6K in $\mathbb{E}[T]$ would therefore mix two quantities of different meaning.

AH-BKZ attains a smaller $\mathbb{E}[T]$ in four of the seven dimensions and a larger one in the remaining three, with the largest gaps at dimensions 124 and 148. In our experiments the wall-clock time of a failed run is close to that of a successful run in the same setting, so $\mathbb{E}[T]$ factorizes as the ratio of the conditional averages of Table 1 divided by the ratio of the success rates. The first factor is stable across dimensions at approximately 0.81 in favor of AH-BKZ, and the variation in Table 4 is therefore driven by the success counts of Table 2. Those counts differ by one to three trials out of ten. The Wilson 95% intervals for the success probabilities of the two methods overlap in every dimension; at dimension 148, for instance, they are $[0.17, 0.69]$ for ProPnjBKZ and $[0.06, 0.51]$ for AH-BKZ. Pooling all seven dimensions gives 36 successes out of 70 for ProPnjBKZ against 29 out of 70 for AH-BKZ, and Fisher’s exact test does not separate them ($p = 0.31$). The present number of trials therefore does not resolve whether the success probabilities of the two methods differ. We do not claim an advantage in $\mathbb{E}[T]$, and the speedup claimed in this paper is the reduction of the wall-clock time among successful runs.