Accelerating the Primal Hybrid Attack against Sparse LWE using GPUs

Ludo N. Pulles[1] Paul Vié[2]

December 16, 2025

[1]CWI, Cryptology Group, Amsterdam, the Netherlands

[2]Télécom Paris, Paris, France

Introduction & Motivation

Context: Post-Quantum Cryptography

- Lattice-based cryptography is fundamental to post-quantum KEMs and signatures
  - CRYSTALS-Kyber, CRYSTALS-Dilithium, Falcon (NIST standards)
- Security based on hardness of **Bounded Distance Decoding (BDD)** in LWE lattices
- **Sparse LWE**: Used in Fully Homomorphic Encryption (FHE) bootstrapping
  - Small & sparse secrets $\Rightarrow$ efficiency gains
  - But: smaller search space $\Rightarrow$ potentially more vulnerable to attacks

The Problem

Current State

- Recent attacks (Salsa, Cool & Cruel) use extensive parallelization on large GPU clusters and Cool & Cruel claims to be *currently the best attack on their benchmark settings*
- But: Lattice estimator (from Martin Albrecht) predicts that theses instances can be broken with modest resources
- However: No efficient open-source implementation exists

Our Goal

Resolve this situation: Implement and accelerate the **Guess + Verify** (G+V) primal hybrid attack using GPUs, validating theoretical predictions in practice

Mathematical Background

Lattices: Definitions

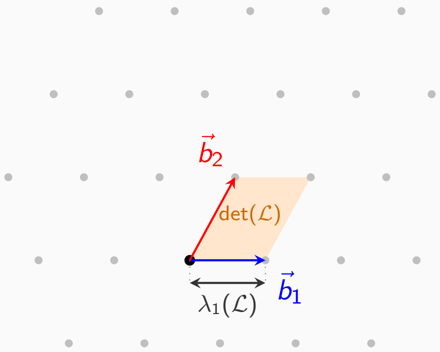
Definition (Lattice)

A lattice $\mathcal{L} \subseteq \mathbb{R}^d$ is a **discrete subgroup** of $\mathbb{R}^d$.
Given a basis $\mathbf{B} = (\vec{b}_1, \ldots, \vec{b}_n)$, it is defined as:

$$\mathcal{L}(\mathbf{B}) = \left\{ \sum_{i=1}^{n} c_i \vec{b}_i \mid c_i \in \mathbb{Z} \right\}$$

Key Invariants

- **Rank**: n (full-rank if $n = d$).
- **Determinant**: $\det(\mathcal{L}) = \sqrt{\det(\mathbf{B}^T \mathbf{B})}$
  (Volume of the fundamental domain)
- **First minimum**: $\lambda_1(\mathcal{L}) = \min_{\vec{v} \in \mathcal{L} \setminus \{\vec{0}\}} \|\vec{v}\|$



Lattice Density: The Gaussian Heuristic

The Intuition

A lattice $\mathcal{L}$ is a discrete grid. How dense is it?

- Denser lattices have shorter vectors.
- Density is inverse to the determinant (volume).

Gaussian Heuristic Prediction

For a random lattice of rank d, the length of the shortest vector $\lambda_1(\mathcal{L})$ is estimated by:

$$\lambda_1(\mathcal{L}) \approx \mathrm{GH}(d) \cdot \det(\mathcal{L})^{1/d} \quad \text{where } \mathrm{GH}(d) \approx \sqrt{\frac{d}{2\pi e}}$$

Orthogonalization: GSO and QR

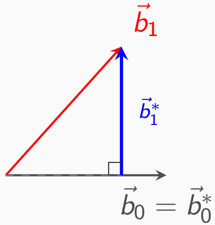
Gram-Schmidt Orthogonalization (GSO)

Constructs an orthogonal basis $(\vec{b}_1^*, \ldots, \vec{b}_n^*)$ from $\mathbf{B}$.

$$\vec{b}_i^* = \vec{b}_i - \sum_{j=0}^{i-1} \mu_{i,j} \vec{b}_j^* \quad (\text{proj.} \ \perp \text{span}(\vec{b}_0 \ldots \vec{b}_{i-1}))$$

where the coefficients are:

$$\mu_{i,j} = \frac{\langle \vec{b}_i, \vec{b}_j^* \rangle}{\|\vec{b}_j^*\|^2}$$



$\vec{b}_1^*$ represents the "height" of $\vec{b}_1$ above $\vec{b}_0$.

Orthogonalization: GSO and QR

Equivalence with QR Decomposition

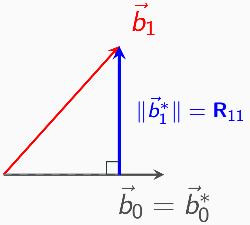
Instead of GSO, we compute $\mathbf{B} = \mathbf{QR}$ where $\mathbf{Q}$ is orthogonal ($\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$) and $\mathbf{R}$ is upper-triangular.

The columns of $\mathbf{Q} = (\vec{q}_1 \ldots \vec{q}_n)$ are the **normalized** GSO vectors:

$$\vec{q}_i = \vec{b}_i^* / \|\vec{b}_i^*\|$$

The matrix $\mathbf{R}$ encodes the GSO geometry:

- **Diagonal:** $\mathbf{R}_{ii} = \|\vec{b}_i^*\|$
- **Off-diagonal:** $\mathbf{R}_{ij} = \langle \vec{b}_j, \vec{q}_i \rangle$
- **GSO Coeffs:** $\mu_{j,i} = \mathbf{R}_{ij} / \mathbf{R}_{ii}$



$\vec{b}_1^*$ represents the "height" of $\vec{b}_1$ above $\vec{b}_0$.

Solving BDD: Babai's Nearest Plane Algorithm

Fundamental Domain

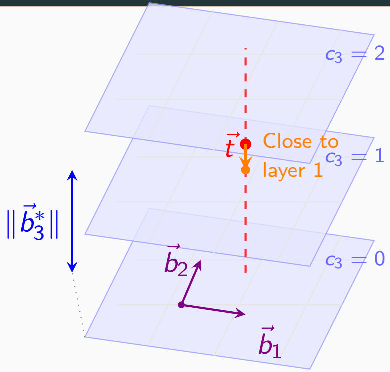
Babai domain of **B**:

$\mathcal{P}(\mathbf{B}^*) = \{\vec{y} \mid \exists \vec{c} \in [-\frac{1}{2}, \frac{1}{2})^n : \vec{y} = \mathbf{B}^*\vec{c}\}$

The Intuition

Babai's algorithm approximates the Closest Vector Problem (CVP).

- It works like solving a triangular system (back-substitution).
- **Difference:** We round to the nearest integer at each step instead of solving exactly.



Output Property

Returns a unique lattice point $\mathbf{B}\vec{c}$ such that the error $\vec{e} \in \mathcal{P}(\mathbf{B}^*)$ (the Babai fundamental domain).

Solving BDD: Babai's Nearest Plane Algorithm

Fundamental Domain

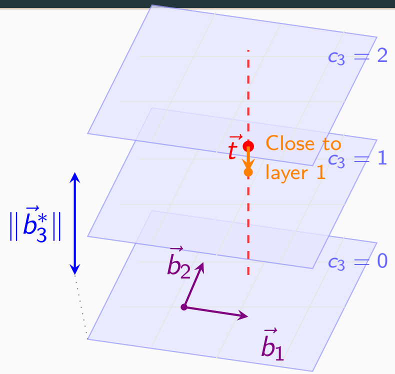
Babai domain of $\mathbf{B}$:

$$\mathcal{P}(\mathbf{B}^*) = \{\vec{y} \mid \exists \vec{c} \in [-\tfrac{1}{2}, \tfrac{1}{2})^n : \vec{y} = \mathbf{B}^* \vec{c}\}$$

Algorithm (Using GSO/QR)

Input: Target $\vec{t}$, Basis $\mathbf{B}$.

1. Write $\vec{t}$ in the GSO basis: $\vec{t} = \sum v_i \vec{b}_i^*$.
2. **Loop** i from n down to 1:
   - $c_i = \lfloor v_i \rceil$ (Nearest integer)
   - Update $\vec{t} \leftarrow \vec{t} - c_i \vec{b}_i$
   - Recompute coords for next step.



Output Property

Returns a unique lattice point $\mathbf{B}\vec{c}$ such that the error $\vec{e} \in \mathcal{P}(\mathbf{B}^*)$ (the Babai fundamental domain).

The Learning with Errors (LWE) Problem

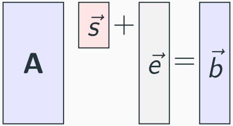
Standard LWE (n, m, q, χ)

Find $\vec{s}$ given $(\mathbf{A}, \vec{b})$ where:

$$\vec{b} = \mathbf{A}\vec{s} + \vec{e} \pmod{q}$$

$$\mathbf{A} \leftarrow \mathcal{U}(\mathbb{Z}_q^{m \times n})$$

$$\vec{s}, \vec{e} \leftarrow \chi \text{ (Gaussian/Binomial)}$$



Sparse LWE (Our Focus)

The secret $\vec{s}$ is **sparse**:

- Hamming weight $h \ll n$.
- Only h non-zero entries.

Distributions:

- **Ternary:** Non-zeros are ± 1.
- **Binomial:** Non-zeros from $\mathcal{B}_\eta$.

0	0	**1**	0	0	$-$**1**	0	0

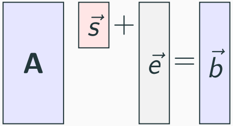
$\leftarrow$ Mostly 0s

The Learning with Errors (LWE) Problem

Standard LWE (n, m, q, χ)

Find $\vec{s}$ given $(\mathbf{A}, \vec{b})$ where:

$$\vec{b} = \mathbf{A}\vec{s} + \vec{e} \pmod q$$
$$\mathbf{A} \leftarrow \mathcal{U}(\mathbb{Z}_q^{m \times n})$$
$$\vec{s}, \vec{e} \leftarrow \chi \text{ (Gaussian/Binomial)}$$



Sparse LWE (Our Focus)

The secret $\vec{s}$ is **sparse**:

- Hamming weight $h \ll n$.
- Only h non-zero entries.

0	0	**1**	0	0	$-$**1**	0	0

$\leftarrow$ Mostly 0s

Security Gap

Search space drastically reduced:

$$|\text{Supp}(\chi)|^n \xrightarrow{\text{Sparse}} \binom{n}{h} \cdot |\text{Supp}(\chi)|^h$$

$\Rightarrow$ Vulnerable to **Hybrid Attacks**.

BDD: Definition and Complexity

Definition (Search-BDD)

Given a basis $\mathbf{B}$ and a target $\vec{t}$ close to the lattice:

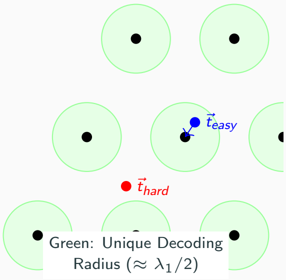
$$\vec{t} = \mathbf{B}\vec{c} + \vec{e}$$

Find the lattice vector $\vec{v} = \mathbf{B}\vec{c}$.

Hardness (Parameter α)

Difficulty depends on error norm $\|\vec{e}\| \approx \alpha \cdot \lambda_1(\mathcal{L})$.

- $\alpha < 1/2$: **Easy.** Unique solution guaranteed.
- $\alpha \in [1/2, 1)$: **Gap.** Solution likely unique.
- $\alpha \geq 1$: **Hard.** Multiple solutions.



BDD becomes hard when the target $\vec{t}$ is outside the packing spheres.

BDD: Definition and Complexity

Definition (Search-BDD)

Given a basis $\mathbf{B}$ and a target $\vec{t}$ close to the lattice:

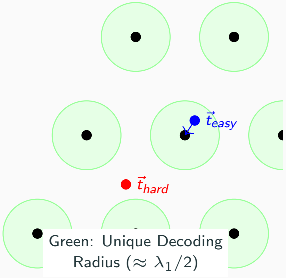
$$\vec{t} = \mathbf{B}\vec{c} + \vec{e}$$

Find the lattice vector $\vec{v} = \mathbf{B}\vec{c}$.

Babai's Success Condition

Finds the solution if error is in the "box":

$$|\langle \vec{b}_i^*, \vec{e} \rangle| \leq \frac{1}{2} \|\vec{b}_i^*\|^2 \quad \forall i$$



BDD becomes hard when the target $\vec{t}$ is outside the packing spheres.

BDD Reduction to Projected Sublattice

Lemma (Reduction Lemma)

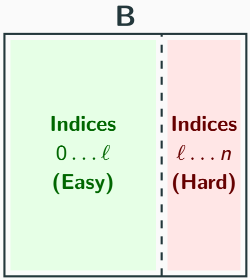
Let $(\mathbf{B}, \vec{t})$ be a BDD instance with error $\vec{e}^{\circ}$. Let $\ell = n - n'$.

The full instance is solved if:

1. **Babai succeeds on the top-left:**

$$|\langle \vec{b}_i^*, \vec{e}^{\circ} \rangle| \leq \frac{1}{2} \|\vec{b}_i^*\|^2 \quad \forall i < \ell$$

2. **Oracle succeeds on bottom-right:** *SearchBDD solves the projected instance correctly.*



BDD Reduction to Projected Sublattice

Algorithm: BDDReduce($\mathbf{B}, \vec{t}, n'$)

Input: Basis $\mathbf{B}$, target $\vec{t}$, block size n'.

$\ell = n - n'$.

1. **Solve Hard Part (Projected):**

$$(\vec{c}_2, \vec{e}') \leftarrow \text{SearchBDD}(\mathbf{B}_{[\ell:n,\ell:n]}, \pi_\ell^\perp(\vec{t}))$$

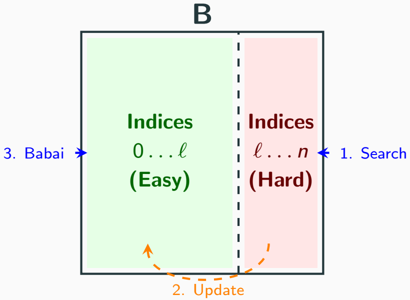
2. **Update Target (Back-Subst):**

$$\vec{t}_{res} = \vec{t} - \mathbf{B}_{[0:n,\ell:n]}\vec{c}_2$$

3. **Solve Easy Part (Babai):**

$$(\vec{c}_1, \vec{e}) \leftarrow \text{BabaiNP}(\mathbf{B}_{[0:\ell,0:\ell]}, \vec{t}_{res})$$

4. **Output:** Return $(\vec{c} = (\vec{c}_1, \vec{c}_2), \vec{e})$



The Primal Attack: Embedding (Bai-Galbraith)

1. The Embedding

Construct a lattice $\Lambda(\mathbf{B})$ containing the error:

$$\underbrace{\begin{pmatrix} \vec{b} \\ 0 \end{pmatrix}}_{\vec{t}} = \mathbf{B} \begin{pmatrix} \vec{u} \\ \vec{s} \end{pmatrix} + \underbrace{\begin{pmatrix} \vec{e} \\ -\xi \vec{s} \end{pmatrix}}_{\vec{e}_{emb}}$$

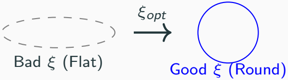
$$\begin{pmatrix} q\mathbf{I}_m & \mathbf{A} \\ \mathbf{0} & \xi \mathbf{I}_n \end{pmatrix} = \mathbf{B}$$

2. Optimization (Bai-Galbraith)

We tune ξ to balance the volume and error norm.

$$\xi = q^{m/n} \frac{\|\vec{e}\|}{\|\vec{s}\|} \sqrt{\frac{n}{m}}$$

$\Rightarrow$ Makes the error $\vec{e}_{emb}$ "spherical" relative to the volume $\det(\mathcal{L})$.



Optimizing ξ aligns the error shape with the lattice geometry, making BDD easier.

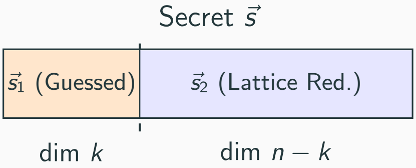
Primal Hybrid Attacks

Primal Hybrid Attack

Concept: Hybrid Splitting

Split the secret $\vec{s}$ into two parts:

- **Guessed Part ($\vec{s}_1$):** Exhaustive search.
- **Unknown Part ($\vec{s}_2$):** Recovered via Lattice Reduction (BDD).



Strategy Trade-off

Context: $\vec{s}$ is sparse.

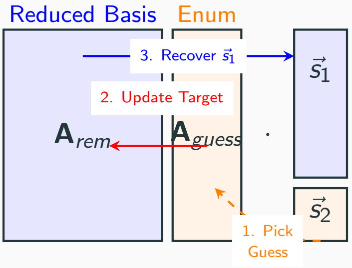
	Drop (Guess 0)	**Full Guess**
Assumption	$\vec{s}_1 = \vec{0}$	$\vec{s}_1 \in$ Support
Cost/Iter	**Low**	High
# Candidates	1	$\approx \binom{k}{h'}$
Success Prob.	Low	**High**
Use Case	Very Sparse	Moderately Sparse

"Drop & Solve" relies on the high probability that indices in $\vec{s}_1$ are zero.

Guess + Verify: The Algorithm

Parameters

- k: Dim. of guessed part (exhaustive).
- h': Weight of guessed part.
- β: BKZ block size (reduction).



Complexity

Cost $\approx \binom{k}{h'} \cdot T_{\text{Babai}} + T_{\text{reduction}}$

Guess + Verify: The Algorithm

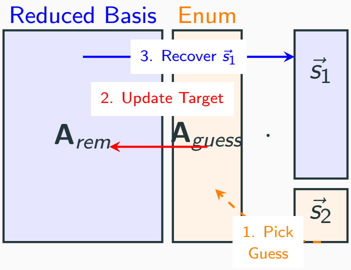
Execution Flow

Preprocessing: Reduce basis $\mathbf{B}$ of dim $n - k$.

Online Phase (Loop):

1. **Guess** a candidate $\vec{s}_2$ (of weight h').
2. **Update** the target (remove guess):
   $\vec{t}' = \vec{t} - \mathbf{A}_{guess}\vec{s}_2$
3. **Solve** BDD on the remaining part:
   $\vec{s}_1 \leftarrow \text{Babai}(\mathbf{B}, \vec{t}')$
4. **Verify:** Is the residual error small?

$$\text{If } \|\vec{t}' - \mathbf{A}_{rem}\vec{s}_1\| \leq R \implies \text{Found!}$$



Complexity

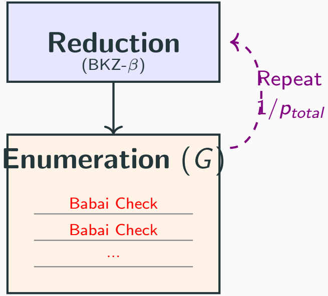
$$\text{Cost} \approx \binom{k}{h'} \cdot T_{\text{Babai}} + T_{\text{reduction}}$$

Complexity Analysis: The Master Equation

Total Attack Cost (Work Factor)

To find the secret with high probability:

$$\text{Cost} \approx \frac{1}{p_{\text{total}}} \cdot (\text{Cost}_{\text{BKZ-}\beta} + G \cdot \text{Cost}_{\text{Babai}})$$



The Components

Probability (p_{total}):

$$p_{\text{comb}}(k, h') \times p_{\text{babai}}(\beta, k, h')$$

(Prob. that guess is in support × Prob. Babai works)

Search Space (G):

$$G \approx \binom{k}{h'} \cdot |\text{Supp}(\chi_s)|^{h'}$$

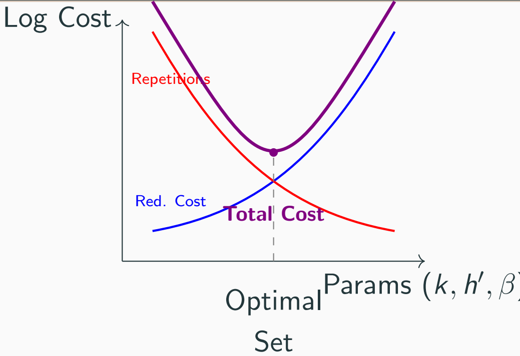
(Number of candidates to test per reduction)

Optimizing the Trio (k, h', β)

The 3 Tuning Knobs

We minimize the cost by balancing:

1. **Blocksize β (Lattice Strength):**
   - $\uparrow$ Probability (Geometry).
   - $\downarrow$ Cost explodes (Exp).
2. **Dimension k (Hybrid Split):**
   - $\uparrow$ Reduction is easier (dim $n - k$).
   - $\downarrow$ Search space G grows.
3. **Guess Weight h' (Coverage):**
   - $\uparrow$ Probability (Combinatorics).
   - $\downarrow$ G explodes $\binom{k}{h'}$.



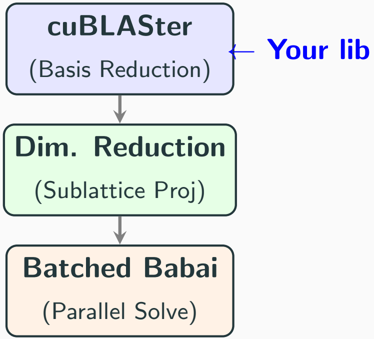
Objective Function

$$\text{Minimize} \approx \frac{\text{Cost}_{BKZ-\beta} + G(k, h') \cdot \text{Cost}_{\text{Babai}}}{p_{\text{total}}(\beta, k, h')}$$

Main Contributions

Our Contributions: A Full GPU Pipeline

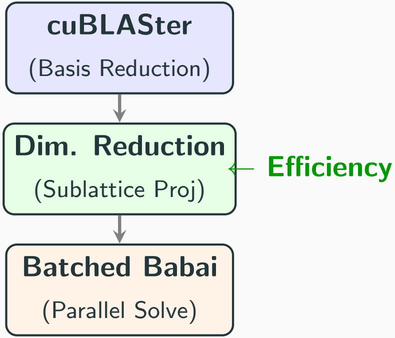
1. **cuBLASter: GPU Lattice Reduction**
   - Port of BLASter (Asiacrypt '25) to CUDA/CuPy.
   - Fast LLL/DeepLLL/BKZ.
   - Bridges gap to GPU-G6K ($\beta \geq 60$).



Our Contributions: A Full GPU Pipeline

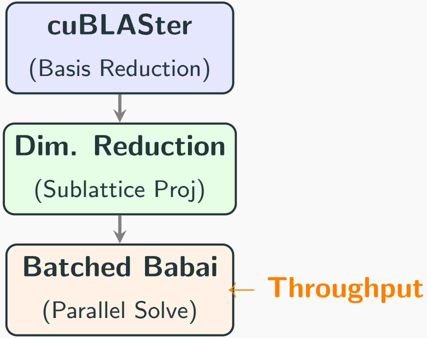
1. **cuBLASter: GPU Lattice Reduction**
   - Port of BLASter (Asiacrypt '25) to CUDA/CuPy.
   - Fast LLL/DeepLLL/BKZ.
   - Bridges gap to GPU-G6K ($\beta \geq 60$).

2. **Smart BDD Preprocessing**
   - **Dimension Reduction:** Project to sublattice before decoding.
   - Accelerates BDD solving.



Our Contributions: A Full GPU Pipeline

1. **cuBLASter: GPU Lattice Reduction**
   - Port of BLASter (Asiacrypt '25) to CUDA/CuPy.
   - Fast LLL/DeepLLL/BKZ.
   - Bridges gap to GPU-G6K ($\beta \geq 60$).
2. **Smart BDD Preprocessing**
   - **Dimension Reduction:** Project to sublattice before decoding.
   - Accelerates BDD solving.
3. **Batched Babai on GPU**
   - Massively parallel verification.
   - Exploits cuBLAS batch operations.



Implementation: Open Source

GitHub Repositories

- **cuBLASter**: `https://github.com/ludopulles/cuBLASter`
- **GPU Primal Hybrid**: `https://github.com/ludopulles/GPUPrimalHybrid`

Key Technologies

- CUDA + CuPy (NumPy-compatible GPU API)
- cuBLAS and cuSOLVER for linear algebra
- Custom GPU kernels for specialized operations
- Integration with fplll and G6K

Why GPUs for Lattice Reduction?

The Bottleneck: GSO Updates

Lattice reduction (LLL, BKZ) is dominated by floating-point arithmetic.

- **Heavy Compute:** Gram-Schmidt Orthogonalization (GSO) costs $O(n^3)$ (by QR).
- **Memory Bound:** Large basis matrices saturate CPU caches.

The Solution: GPU Offloading

GPUs offer massive throughput for linear algebra.

- **Massive Parallelism:** Thousands of cores to parallelize independent operations.
- **Specialized Hardware/Libs:** Optimized for matrix multiplications, triangular solves, QR factorizations.

The Key Constraint: Batching

To leverage GPUs effectively, we must batch small linear algebra operations to amortize kernel launch overhead.

Seysen's Size Reduction Algorithm

Classical Approach (CPU)

Recursive algorithm on upper-triangular $\mathbf{R} \in \mathbb{R}^{n\times n}$:

1. Split: $\mathbf{R} = \begin{pmatrix} \mathbf{R}_{11} & \mathbf{R}_{12} \\ 0 & \mathbf{R}_{22} \end{pmatrix}$ with $\mathbf{R}_{11}$ size $\lfloor n/2 \rfloor$
2. Recursively reduce $\mathbf{R}_{11}$ and $\mathbf{R}_{22}$
3. Reduce $\mathbf{R}_{12}$ with respect to $\mathbf{R}_{11}$

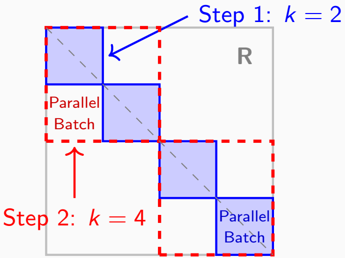
Problem: Sequential recursion doesn't parallelize well, because R11 and R22 may be not the same size. (If n is not a power of two)

Batched Size Reduction: Motivation

GPU Optimization: Batched Reduction

Key insight: Many submatrices of the same size can be reduced in parallel

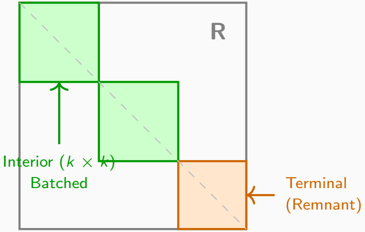
- Parse **R** with $\mathbf{R}_{11}$ size $\frac{1}{2} \cdot 2^{\lceil \log_2(n) \rceil}$ (next power of 2)
- For each level $k = 2, 4, 8, \ldots, 2^{\lceil \log_2(n) \rceil}$:
  - Identify all $k \times k$ diagonal blocks: $\mathbf{R}_{[ik:(i+1)k, ik:(i+1)k]}$
  - Reduce all blocks in one batched kernel call
- Total: $2\lceil \log_2 n \rceil$ kernel sequences (vs $O(n)$ sequential calls)



Batched Size Reduction: Implementation Details

Interior vs Terminal Blocks

- **Interior blocks**: Full $k \times k$ submatrices, can batch efficiently
- **Terminal block**: Last block may be smaller $(n - \lfloor n/k \rfloor k) \times (n - \lfloor n/k \rfloor k)$, handled separately



Optimizations

- Level $k = 2$: Vectorized super-diagonal update (single kernel)
- For $k < 4$ interior blocks: Fall back to unbatched path (avoid launch overhead)

Performance Gain

Batching reduces kernel launch overhead from $O(n)$ to $O(\log n)$ calls

Theoretical Foundation: Recursive Batch Nearest Plane (BLASter)

Algorithm 1 BatchNearestPlane($\mathbf{R}, \mathbf{T}$)

Input: $\mathbf{R} \in \mathbb{R}^{n \times n}, \mathbf{T} \in \mathbb{R}^{n \times N}$
if $n = 1$ **then**
  $\mathbf{C} \leftarrow \lfloor \mathbf{T} / \mathbf{R} \rceil$
  $\mathbf{T} \leftarrow \mathbf{T} - \mathbf{RC}$
  return $\mathbf{C}$
else
  Split $\mathbf{R} = \begin{pmatrix} \mathbf{R}_{11} & \mathbf{R}_{12} \\ 0 & \mathbf{R}_{22} \end{pmatrix}, \mathbf{T} = \begin{pmatrix} \mathbf{T}_1 \\ \mathbf{T}_2 \end{pmatrix}$
  $\mathbf{C}_2 \leftarrow \text{BatchNP}(\mathbf{R}_{22}, \mathbf{T}_2)$ {Recurse Bottom}
  $\mathbf{T}_1 \leftarrow \mathbf{T}_1 - \mathbf{R}_{12}\mathbf{C}_2$ {Update Top}
  $\mathbf{C}_1 \leftarrow \text{BatchNP}(\mathbf{R}_{11}, \mathbf{T}_1)$ {Recurse Top}
  return $\binom{\mathbf{C}_1}{\mathbf{C}_2}$
end if

Complexity

By using fast matrix multiplication for the update step, the runtime is:

$$O(N \cdot n^{\omega - 1})$$

Batched Nearest Plane: The Blocked Algorithm (BLASter)

The Core Idea

We simulate the recursive calls $R_{[i,k)} \to (R_{[i,j)}, R_{[j,k)})$. Instead of updating the whole matrix at each step, we **wait** until the bottom block $[j,k)$ is fully solved to update the top block $[i,j)$ in one go.

The Procedure (Iterate j from n down to 1):

1. **Scale & Round:** $\vec{c}_j = \left\lfloor \frac{1}{R_{jj}} \cdot \mathbf{T}_{j,\dots} \right\rceil \in \mathbb{Z}^N$
2. **Lazy Update (Based on Recursion Structure):** We check if j is the "split point" of a recursion block $[i,k)$.
   - **If yes:** We have all coefficients $\mathbf{C}_{j:k}$ ready.
   - We perform the update corresponding to the $R_{12}C_2$ term in the algorithm:

$$\mathbf{T}_{i:j} \leftarrow \mathbf{T}_{i:j} - \mathbf{R}_{i:j,\ j:k} \times \mathbf{C}_{j:k}$$

(Otherwise, we just do a minimal scalar update to prepare row $j-1$)

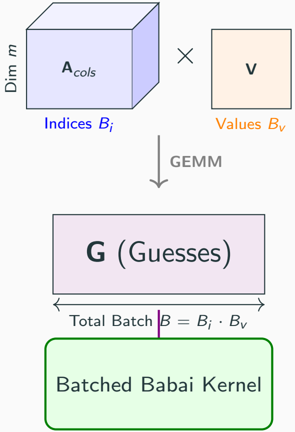
Ideally suited for GPU: Since N is huge, the update step becomes a massive matrix operation.

Batched Babai: The GPU Pipeline

1. Batch Structure ($B = B_i \cdot B_v$)

We split the search space into:

- **Indices (B_i):** Choice of support S in the guessed part ($B_i \leq \binom{k}{3}$), typically $1024 \ldots 4096$, to fit in GPU memory).
- **Values (B_v):** Coefficients on fixed S. (e.g., for $\chi_s \leftarrow \mathcal{B}_2$ and $h' = 3$, $B_v = 4^3 = 64$)



Batched Babai: The GPU Pipeline

2. Execution Flow (5 Steps)

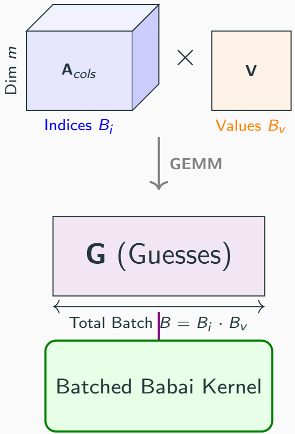
1. **Gather:** Fetch columns of **A** into tensor $\mathcal{A} \in \mathbb{R}^{B_i \times m \times h'}$.
2. **Stack Values:** Prepare $\mathbf{V} \in \mathbb{R}^{h' \times B_v}$.
3. **Batch GEMM:** Compute guesses.

   $$\mathbf{G} \leftarrow \text{Reshape}(\mathcal{A}) \times \mathbf{V}$$

   Result $\mathbf{G}$ is $m \times (B_i \cdot B_v)$.
4. **Project:** Update targets in parallel.

   $$\mathbf{T} = \mathbf{Q}_2^\top (\vec{b}\vec{1}^\top - \mathbf{G})$$

5. **Solve:** Run Babai on $(\mathbf{R}_{22}, \mathbf{T})$.

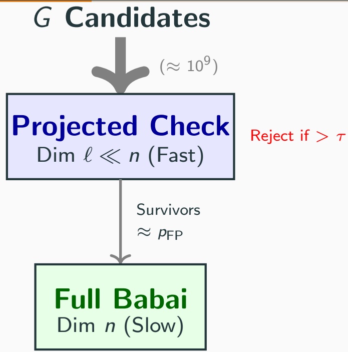


Dimension Reduction: The Pre-Filtering Strategy

Motivation: fast-reject

Full Babai is expensive ($O(n^2)$). We need a cheap pre-filter.

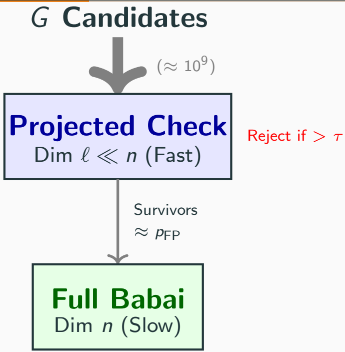
- **Idea:** Project lattice to last ℓ coordinates.
- **Check:** If $\|\pi_\ell(\vec{e})\| > \tau$, reject immediately.



Trade-off

Small $\ell \to$ Faster check but requires tighter τ (risk of False Negatives).

Dimension Reduction: The Pre-Filtering Strategy



False Positive Rate (p_{FP})

We model the target as uniform random in the projected torus. The expected number of false positives is:

$$\mathbb{E}[\text{FP}] \approx G \cdot \frac{\text{Vol}(\tau \mathcal{B}_\ell)}{\det(\mathbf{R}_{\text{proj}})}$$

To cap FP at 1%, we set τ such that:

$$\tau \approx \text{GH}(\ell) \cdot \left(\frac{0.01 \cdot \det(\mathbf{R}_{\text{proj}})}{G} \right)^{1/\ell}$$

Trade-off

Small $\ell \rightarrow$ Faster check but requires tighter τ (risk of False Negatives).

Dimension Reduction: True Positive Analysis

The "Good" Candidate

For the correct guess, the residual is Gaussian noise:

$$\vec{e}_{\text{proj}} \sim \mathcal{N}(0, \sigma^2 \mathbf{I}_\ell)$$

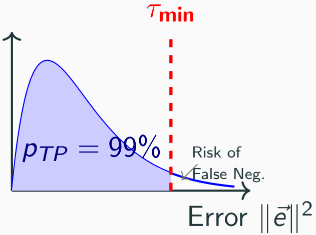
Its squared norm follows a Chi-squared distribution:

$$\|\vec{e}_{\text{proj}}\|^2/\sigma^2 \sim \chi^2_\ell$$

The Constraint (Lower Bound)

We must accept the correct guess with prob p_{TP} (e.g., 99%):

$$\tau_{\text{proj}} \geq \sigma \cdot \sqrt{\text{Quantile}_{\chi^2_\ell}(p_{TP})}$$



Synthesis

We need a gap between this τ_{min} (TP) and the previous τ_{max} (FP).

Dimension Reduction: Finding the Optimal ℓ

The Feasibility Condition

We need a radius τ that satisfies both:

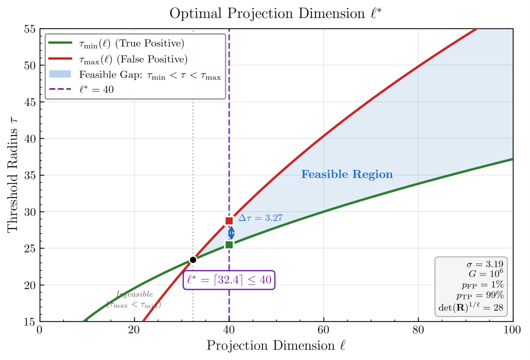
1. **High enough** to catch the secret (TP).
2. **Low enough** to filter bad guesses (FP).

Selection Algorithm

Find the **smallest** ℓ such that:

$$\underbrace{\sigma\sqrt{q_{\chi^2_\ell}(p_{\mathrm{TP}})}}_{\text{Min } \tau \text{ (Noise)}} < \underbrace{\mathrm{GH}(\ell)\cdot\left(\frac{p_{\mathrm{FP}}\det|\mathbf{R}|}{G}\right)^{1/\ell}}_{\text{Max } \tau \text{ (Density)}}$$

Then pick τ in the gap.



Benefit

Reduces verification cost from $O(n^2)$ to $O((\ell^*)^2)$. Typically $\ell^* \ll n$ (e.g., 40 vs 1000).

Experimental Results

Experimental Setup

Hardware Platforms

Machine	CPU	Cores	GPU
Y	AMD EPYC-Milan 2.745GHz	96	NVIDIA H100 (1)
H	Intel Xeon Gold 6248 2.5GHz	80	NVIDIA RTX 2080 Ti (4)
Z	Intel Xeon Gold 5222 3.8GHz	16	NVIDIA RTX 2080 Ti (8)

Comparison Baseline

Cool & Cruel benchmarks :

- Large computing cluster with 256 NVIDIA V100 GPUs
- Report: minimum GPU-hours over 10 runs (when successful)

Fair Comparison

Focus on **GPU-hours** and **core-hours**, not wall time (accounts for parallelism)

Experimental Protocol: LWE Instances

LWE Parameter Sets

Extremely sparse instances from Wenger et al.:

Parameter	Set A	Set B
Error Dist.	Binomial	Gaussian
Std Dev	$\eta = 2$	$\sigma = 3.19$
Secret h	$h \in \{9, \ldots, 25\}$	
Sparsity	Fixed Hamming Weight	
Dimension	$n = \kappa \cdot D \in \{512, 1024\}$	
Structure	Module-LWE ($\kappa \in \{1, 2\}$)	

Reproducibility

- **Sample Size:** 5-6 instances per set.
- **Control:** Fixed PRNG seed for deterministic generation.

Robust Metric

Unlike some benchmarks that report the *minimum* time (lucky runs), we report:

Average Wall-Time
(over successful runs)

Success Rate notation: x/y (x successes / y attempts).

Lattice Reduction Benchmarks: cuBLASter vs BLASter

	Dimension		Wall time		
Algorithm	512	1024	flatter	BLASter	cuBLASter
LLL	512	—	156 s	6.8 s	**2.9 s**
LLL	—	1024	—	26.0 s	**5.6 s**
BKZ-60	512	—	—	363 s	**218 s**
DeepLLL-4	—	1024	904 s	223 s	**49 s**

- cuBLASter outperforms BLASter for $n \geq 512$
- $\mathbf{2} - \mathbf{4}\times$ speedup for large dimensions
- Progressive BKZ-60 in dim 512: 40% faster

Attack Success: Guess + Verify vs Cool & Cruel

Instance			Parameters		G+V (ours)		C+C	
Type	n	h	k	h'	succ.	GPU-h	succ.	GPU-h
Bin	$2 \cdot 256$	11	393	3	**5/6**	**5.7**	2/10	26 ± 13
Bin	$2 \cdot 256$	12	395	3	**4/5**	**30.1**	—	—
Bin	$2 \cdot 256$	20	234	≤ 3	**4/5**	**22.6**	3/10	51 ± 13
Bin	$2 \cdot 256$	21	235	≤ 3	**5/5**	**27.9**	3/10	154 ± 13
Bin	$2 \cdot 256$	25	235	≤ 3	**5/5**	**164.0**	1/10	10752 ± 128
Ter	$1 \cdot 1024$	11	768	3	**5/5**	**9.1**	1/10	102 ± 52
Ter	$1 \cdot 1024$	9	800	3	5/5	**9.4**	10/10	31 ± 6
Ter	$1 \cdot 1024$	10	801	3	**5/5**	**42.3**	0/10	—

Performance Comparison

Guess + Verify Advantages

- G+V achieves **higher success rates** on almost all instances
- G+V solves instances where C+C fails (e.g., Ter with $h = 10$)
- **Lower GPU utilization** than C+C (even including lattice reduction!)

Hardware

- Our experiments: 1 NVIDIA H100 or 2-8 RTX 2080 Ti GPUs
- C+C benchmark: Large cluster with 256 NVIDIA V100 GPUs
- Fair comparison: Focus on core-hours and GPU-hours, not wall time

Conclusion

Summary

Contributions

1. **cuBLASter**: Fast GPU lattice reduction library
   - 2-4× speedup over BLASter for $n \geq 512$
2. **Efficient Guess + Verify implementation**
   - Dimension reduction for BDD
   - Batched Babai's Nearest Plane on GPU
3. **Practical validation** of primal hybrid attacks
   - Outperforms Cool & Cruel in success rate and efficiency
   - Open-source baseline for sparse LWE attacks

Impact & Future Work

Security Implications

- Practical demonstration that primal hybrid attacks are effective against sparse LWE
- Validates lattice estimator predictions

Future Directions

- Explore meet-in-the-middle in primal hybrid guessing
- Optimize cuBLASter for even larger dimensions (Implement CUDA enumeration, etc.)
- Explore other usecases of cuBLASter and batched Babai NP in lattice-based cryptanalysis

Thank you!

Questions?

Code available at:

github.com/ludopulles/cuBLASter

github.com/ludopulles/GPUPrimalHybrid

Preprints:

https://ia.cr/2025/1990

https://ia.cr/2025/1002

Reference for benchmarks of Cool & Cruel

E. Wenger, E. Saxena, M. Malhou, E. Thieu and K. Lauter, "Benchmarking Attacks on Learning with Errors", in *S&P 2025*
url: https://ieeexplore.ieee.org/document/11023470