p3Enum: A new Parameterizable and Shared-Memory Parallelized Shortest Vector Problem Solver

Michael Burger, Christian Bischof, and Juliane Krämer

Fachbereich Informatik
Technische Universität Darmstadt, Hochschulstraße, 10, 64289 Darmstadt, DE
{michael.burger,christian.bischof}@sc.tu-darmstadt.de,
jkraemer@cdc.informatik.tu-darmstadt.de

Abstract. Due to the advent of quantum computers, quantum-safe cryptographic alternatives are required. Promising candidates are based on lattices. The hardness of the underlying problems must also be assessed on classical hardware. In this paper, we present the open source framework p3Enum for solving the important lattice problem of finding the shortest non-zero vector in a lattice, based on enumeration with extreme pruning. Our parallelized enumeration routine scales very well on SMP systems with an extremely high parallel efficiency up to 0.91 with 60 threads on a single node. A novel parameter v within the pruning function increases the probability of success and the workload of the enumeration. This enables p3Enum to achieve runtimes for parallel enumerations which are comparable to single-threaded cases but with higher success rate. We compare the performance of p3Enum to publicly available libraries and results in the literature. For lattice dimensions 66 to 88, p3Enum performs the best which makes it a good building block in lattice reduction frameworks.

1 Introduction

We need cryptography in our daily lives to secure the applications like social media and online banking. However, we know that all public-key cryptography based on prime factorization and elliptic curves in use today will be broken once large-scale quantum computers exist. Therefore, a vivid field of research is post quantum cryptography, where cryptographic algorithms that withstand attacks with quantum computers are developed. Algorithms based on lattices are promising since they are versatile and efficient. One of the frequently studied algorithmic problems in lattice cryptography is the shortest vector problem (SVP) which cannot be solved by quantum computers efficiently. However, to be practical, lattice-based cryptography does not only have to withstand attacks with quantum computers, but the exact hardness in relation to classical computers and HPC systems has also to be determined so that secure parameters, e.g., key sizes, can be chosen. Regarding the SVP, the most promising techniques to solve it are extreme pruning [2] and sieving [4]. To understand their full potential, they have to be analyzed on parallel systems. To that end, we present p3Enum which is a **p**arallelized, **p**arameterizable open source framework based on extreme **p**runing.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_48

2 Preliminaries

We denote vectors with bold lower case letters, e.g., $\mathbf{u}$, matrices with bold upper case letters, e.g., $\mathbf{B}$, and scalars with normal lower case letters, e.g., β. $\mathbf{B}^{m\times n}$ stands for an $m \times n$ matrix. If the dimensions are clear from the context, we simply write $\mathbf{B}$. Integers are denoted by $\mathbb{Z}$ and the real numbers by $\mathbb{R}$. The standard inner product is denoted by $\langle\cdot,\cdot\rangle$ and the Euclidean norm by $\|\cdot\|$.

A lattice of dimension d is a discrete additive subgroup of $\mathbb{R}^d$. Every lattice $\Lambda \subset \mathbb{R}^d$ can be represented by a basis, i.e., a set of $\mathbb{R}$-linearly independent vectors $\mathbf{B} = \{\mathbf{b}_1,\ldots,\mathbf{b}_n\} \subset \mathbb{R}^d$ such that $\Lambda = \Lambda(\mathbf{B}) = \mathbb{Z}\mathbf{b}_1 + \cdots + \mathbb{Z}\mathbf{b}_n$. We identify lattice bases with matrices whose columns represent the basis vectors. In this case, d is called the dimension of the lattice and $n \leq d$ is called its rank. If $n = d$, the lattice is called a full-rank lattice. All lattices within this work are full-rank. The Gram-Schmidt (GS) basis (obtained by GS orthogonalization) of a basis $\mathbf{B}$ is denoted by $\mathbf{B}^* = \{\mathbf{b}_1^*,\ldots,\mathbf{b}_n^*\} \subset \mathbb{R}^d$, the respective GS-lengths by $||\mathbf{b}_1^*||^2,\ldots,||\mathbf{b}_n^*||^2$, and the GS-coefficients by $\mu_{i,j}$ with $1 \leq j < i \leq n$.

The quality of a lattice basis $\mathbf{B}$ can, e.g., be measured by the decrease of the series $||\mathbf{b}_1^*||^2,\ldots,||\mathbf{b}_n^*||^2$, or by the value of $||\mathbf{b}_1^*||^2$. Improving the quality of a basis is called *basis reduction*. Geometrically, basis reduction means, in particular, to make the basis vectors shorter and more orthogonal. The most commonly used basis reduction algorithm is BKZ 2.0 [2]. BKZ 2.0 works on local blocks of lattices of dimension $\beta < n$ and optimizes the basis by sliding over all basis vectors in contiguous blocks. A basis processed by BKZ with block size β is called BKZ-β reduced. Solutions for the SVP or approximately good solutions, delivered by a so-called SVP-oracle, are required within each local block. The two most common types of SVP-oracles employ enumeration [2, 3, 5] with extreme pruning or sieving algorithms [1, 4]. p3Enum uses enumeration and hence searches for coefficient vectors $\mathbf{u}$ fulfilling $||\mathbf{u} * \mathbf{B}|| < \bar{A}$ in a heuristically pruned search tree. The extreme pruning function $\mathbf{A} = (A_1,\ldots,A_n)$ with $A_1 \leq A_2 \leq \cdots \leq A_n \leq \bar{A}$ determines the maximal costs A_i for a partial solution vector with length i. Large parts of the search tree are cut off and in general the search has to be repeated several times on randomized input bases to succeed [2].

3 Related Work

The single-threaded, template-based fplll C++-library [3] implements important algorithms from the lattice domain like LLL, BKZ, or enumeration with extreme pruning. Pruned enumeration is possible for bases with $n \leq 90$.

Kuo et al. [5] presented an implementation of extreme pruning on GPUs. The enumeration tree is split into starting vectors generated on the CPU which are completed on GPUs. The pruning function $\mathbf{A}$ is a scaled polynomial of degree eight.

Aono et al. [2] developed progressive BKZ which avoids a predefined BKZ-strategy but starts with a small β and iteratively increases it in appropriate steps. The pruning functions are based on the so-called full enumeration cost (FEC) which results from benchmarks, heuristics, and optimized estimates for $\bar{A}$.

Concerning sieving, SubSieve [4] employs progressive sieving [6] which works on sublattices instead of directly solving the SVP on the whole lattice basis. It also takes

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_48

advantage of the fact that the output of sieving a list of short vectors. This allows to solve the n-dimensional SVP with sieving calls on $(n-\delta)$-dimensional sublattices, where δ is heuristically determined.

Albrecht et. al [1] combine the principles of SubSieve with further algorithmic improvements into the General Sieve Kernel (G6K). G6K processes the basis in non-contiguous blocks and its parallelized C++-implementation holds the record in the Darmstadt SVP challenge[1] (D-SVPC), where researchers are invited to search short vectors within provided random lattices.

4 Implementation of p3Enum

p3Enum is implemented in C++11[2]. The randomized bases are reduced by two different BKZ calls to the fplll library. First, we execute a classical BKZ without pruned enumeration calls on a relatively small block size of pre-$\beta \in \{2,\ldots,36\}$. It is followed by a call with $\beta \in \{2,\ldots,54\}$ and pruning and heuristical early abortion enabled so that BKZ 2.0 terminates when the heuristic detects no further considerable improvements in the basis quality. The values of pre-β and β depend on the dimension of the lattice and we try to weight the required runtimes in a relation of $2:1$ since our experiments showed the best performance for this combination.

For the extreme pruning function, we employ the polynomial of degree eight from [5] (cf. Section 3) scaled to the respective dimension of the lattice. We evaluate the polynomial at each position $l \in \{1,\ldots,n\}$ and multiply the result $\in]0,\ldots,1.0]$ with $\bar{A}$. This value is assigned to the respective entry A_l of the pruning function vector $\mathbf{A}$.

4.1 Parallelization Strategy

Our parallelization strategy is twofold. First, the enumeration itself is parallelized and second, multiple instances of basis reduction are executed in parallel.

The parallelization of the enumeration is strongly related to the approaches of [2, 5]. Within the OpenMP parallel region, the first thread arriving starts at a pre-defined depth $\eta = 10$ which is chosen based on experiments. This thread enumerates vectors from η to the root which are smaller than the corresponding A_i's, called *candidates*. They are inserted in a thread-safe, shared ring-queue, developed by us. The size can be reconfigured at runtime and the fixed size allows to calculate its memory requirements. If the queue is filled above a given threshold, all threads except the one filling the queue start processing the partial tree at level $\eta+1$.

Experiments with the fplll library showed that running multiple instances on the same compute node does not have a considerable negative effect on the runtime compared to running one instance. Hence, p3Enum performs multiple instances on different randomized bases in parallel to make efficient use of the compute capabilities of modern computers. This also considerably reduces, if not even removes, the drawback that serial BKZ-implementations prevent efficient parallelization on an SMP-system, as men-

[1] https://www.latticechallenge.org/svp-challenge/
[2] https://github.com/MiBu84/p3enum

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_48

tioned in [5]. In that way, we create a bunch of randomized, reduced bases for processing in about the same time as a single basis otherwise. One drawback of this approach is that the time to reduce a basis varies for randomized instances. Consequently, some threads finish faster than others. Empirical experiments show a difference of about 2 in the runtime between the fastest and the slowest thread.

4.2 Parameterized Workload

Performing experiments with the progressive BKZ library [2] show a conspicuous behavior. The library internally decides whether to execute the pruned enumeration in the OpenMP-parallelized or the single-threaded version. For many dimensions < 85, the heuristic chooses the single-threaded version since the workload is too small and the predicted single-threaded execution time far below $1\,s$. Comparably fast runtimes can be achieved with p3Enum: When directly running pruned enumeration on a random lattice of dimension 80, which is BKZ-30 reduced and a tight bound $\bar{A}$ for the shortest vector is known (as described below), the single-threaded execution time is below $0.01\,s$ and a parallelization will not pay off. Hence, we introduce a new pruning-parameter ν which works in the following way. We first evaluate the polynomial of [5] for the considered entry $l \in \{1,\ldots,n\}$, resulting in a value $\alpha \in]0,\ldots,1.0]$. Now, we update $\alpha = \min(\alpha + \nu, 1.0)$ and finally calculate the value of the pruning function $\mathbf{A}$ at entry l by $A_l = \alpha \cdot \bar{A}$. Mathematically, we shift the graph of the pruning function along the positive y-axis.

In that way, we increase the probability of keeping the shortest vector in the pruned tree and increase the workload so that a parallelization pays off. The goal is to set ν such that it enables p3Enum to efficiently run parallel enumerations in the time of single-threaded enumerations, but with increased success rate.

4.3 Heuristics to Improve the Performance

To reduce the number of randomized bases to be processed, the bases reduced in parallel are processed in ascending order of their $||\mathbf{b}_1^*||^2$ values. Mainly for smaller dimensions < 80 this reduces the number of processed bases at no additional cost.

To cope with the different runtimes of parallel BKZ-instances, the program measures the runtimes during the first round of reductions and, based on the observed timing variance between threads (cf. Section 4.1), sets a time limit for the BKZ-calls in the following rounds to 2.1 times of the fastest reduction in the first round. If the $||\mathbf{b}_1^*||^2$ value of the BKZ calls terminated because of this time limit lies in the range of the $||\mathbf{b}_1^*||^2$'s of the other bases, the basis is processed normally, otherwise it is discarded.

5 Experiments and Results

5.1 Methodology

system1 nodes are dual socket Intel E5-2680 v3 CPUs (24 cores) with 64 GB of RAM. *system2* nodes are quad socket Intel E7-4890 v2 CPUs (60 cores) with 1024 GB of

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_48

RAM. For p3Enum and fplll we use gcc 8.2.0 and for SubSieve gcc 4.9.4. All random lattices are in Goldstein-Mayer form of the D-SVPC with seeds (0, 237, 6880, 97575, 98937). To have an upper bound for the length of the vectors, we performed, like [4], several runs of fplll's pruned SVP-routine with a target success probability of 99%.

5.2 Performance Analysis

The parameterization for p3Enum ($v \in [0.03,\ldots,0.3], \beta \in \{2,\ldots,58\}$, pre-$\beta \in \{2,\ldots,38\}$) was based on empirical test runs and constant for a lattice dimension d. It is available on p3Enum's github. Figure 1 summarizes the runtimes and splits the data into three diagrams to refine the logarithmic scale on the y-axis. p3Enum, SubSieve, and fplll are visualized as dots which are created by the average of at least 75 measurement points (at least 15 per seed). Finally, the stars indicate the average runtime of G6K whose runtime are taken from [1].

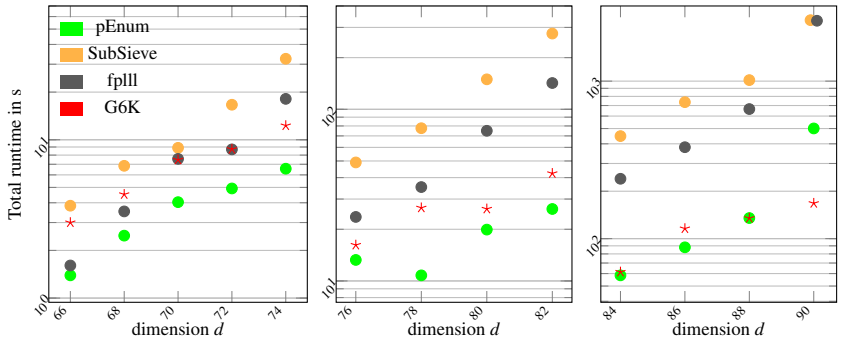

Fig. 1. Performance comparison for dimensions 76-90 on *system1* nodes.

Figure 1 shows that p3Enum delivers faster runtimes than fplll and SubSieve for all tested dimensions. Table 1 summarizes the speedup of p3Enum compared to other solutions.

Table 1. Speedup of p3Enum when compared to other solutions.

solver \ ***n***	66	68	70	72	74	76	78	80	82	84	86	88	90
fplll [3]	1.5	1.4	1.9	1.8	2.8	1.7	3.3	3.8	5.4	4.1	3.7	4.9	4.8
GPUEnum [5]	-	-	-	-	-	*2.4*	*4.8*	4.5	4.2	3.8	2.9	3.6	1.7
ProSieve [6]	*15.7*	*21.9*	29.3	*42.9*	*58.7*	*50.3*	*113*	109	*148*	-	-	-	-
SubSieve [4]	1.6	2.3	2.2	3.4	5.0	3.6	7.2	7.5	10.5	7.7	7.2	7.5	4.9
G6K [1]	1.5	1.5	1.9	1.8	1.9	1.2	2.5	1.3	1.6	1.1	1.1	1.0	0.34

We calculated the speedup when employing p3Enum compared to the implementation in the first column. The values in italics result from least squares fitting and

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_48

extrapolation of the other data points given in the literature. p3Enum outperforms the GPU-enumeration from [5] in all dimensions considered, although [5] employs a system with eight GPUs, providing more theoretical FLOPs and consuming much more energy than *system1*. Progressive sieving [6] is also slower in all dimensions. G6K, however, is faster than p3Enum in dimension 90. This seems to underpin the assumption of [4] that a good implementation of the novel sieving algorithms will outperform all pruned enumeration solvers somewhere around dimension 90. Since p3Enum delivers the best performance in $n = \{66, \ldots 88\}$ it is a very suitable SVP-oracle within basis reduction for that range.

5.3 Scaling Behavior and Efficiency

On our 60-core *system2* we measured the parallel efficiency by the fraction of the achieved speedup compared to the single-threaded baseline with four different lattice dimensions (seed 0). The enumeration was configured such the 60 core runs required between 5 and 10 seconds by setting v. The target length was set shorter than the shortest vector in the lattices to have reproducible runtimes. Figure 2 summarizes the results.

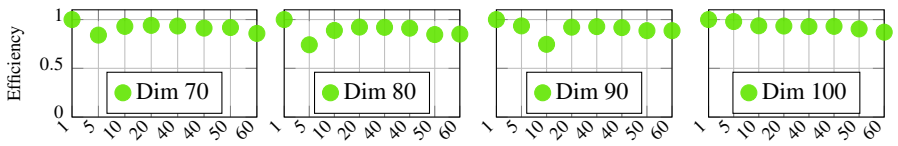

Fig. 2. Parallel efficiency on *system2*. x-axis: # threads employed.

The efficiency with 60 threads is at least 0.85, demonstrating very good scaling. For dimension 100, the efficiency is even higher than 0.9 for 60 threads. In dimensions 80 and 90 we see two outliers at 5 threads and 10 threads, respectively. Since we noticed this effect only for a small number of threads and the efficiency is still over 0.75, we do not consider this a relevant shortcoming of p3Enum.

5.4 New Shortest Vectors in Darmstadt SVP Challenge

We found shorter solutions with a different seed in higher, already solved dimensions for in the D-SVPC. Table 2 compares the fastest succeeding trial of p3Enum (row 1) and SubSieve (row 3), the runtimes of former solutions in the D-SVPC (row 2), and the average time of G6K from [1]. The underlying parameterization for p3Enum varies and may not be the ideal one (cf. D-SVPC entries for details).

Table 2 shows that higher dimensions can be solved with a competitive runtime. p3Enum's time for dimensions 91, 95, and 97 is smaller than the shortest run of fplll and SubSieve achieved on the 90-dimensional bases. Additionally, the results highlight the randomness in the runtime of SubSieve in higher dimensions. Although we performed five runs, the fastest runtime in dimension 94 is considerably lower than in dimension 93. The seeds for G6K were different and no direct comparison is possible.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_48

Table 2. Comparison of runtime in s for higher dimensions.

solver \ *n*	91	92	93	94	95	96	97	98	99	100
p3Enum	25	397	233	279	76	190	65	107	897	4000
D-SVPC	$4.4 \cdot 10^5$	$3.7 \cdot 10^5$		3600	1800	5400			7200	27360
SubSieve	553	823	1642	1015	2083	1447	2872	1734	3467	3363
G6K	-	312	-	375	-	815	-	995	-	1964

6 Conclusion

We introduced the open source framework p3Enum for solving the SVP with its additional parameter ν enabling a parallel efficiency rate of more than 0.9 on a 60-core system by adjusting the workload and the success probability. p3Enum is the fastest solver in dimensions 66-88 compared to available SVP solutions. Hence, p3Enum can be employed as a building block in lattice reduction frameworks. To further increase p3Enum's performance we will extend it with MPI and implement a search for the pruning function as realized in [2, 3].

Acknowledgments

This work has been co-funded by the DFG through CRC 1119 CROSSING and BI 714/6-1. Calculations were conducted on the Lichtenberg computer of the TU Darmstadt, and computing resources granted by RWTH Aachen University under project prep0016. We thank L. Ducas et al. for providing the preliminary version of [1].

References

1. M. Albrecht, L. Ducas, G. Herold, E. Kirshanova, E. W. Postlethwaite, and M. Stevens. The general sieve kernel and new records in lattice reduction. Cryptology ePrint Archive, Report 2019/089, 2019.
2. Y. Aono, Y. Wang, T. Hayashi, and T. Takagi. Improved progressive BKZ algorithms and their precise cost estimation by sharp simulator. In M. Fischlin and J.-S Coron, editors, *Advances in Cryptology - EUROCRYPT 2016*, pages 789–819. Springer, 2016.
3. The fplll development team. fplll, a lattice reduction library. Available at `https://github.com/fplll/fplll`, 2016.
4. L. Ducas. Shortest vector from lattice sieving: A few dimensions for free. In *Advances in Cryptology - EUROCRYPT 2018*, pages 125–145, 2018.
5. P.-C. Kuo, M. Schneider, Ö. Dagdelen, J. Reichelt, J. Buchmann, C.-M. Cheng, and B.-Y. Yang. Extreme enumeration on GPU and in clouds. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011*, pages 176–191. Springer, 2011.
6. T. Laarhoven and A. Mariano. Progressive lattice sieving. In *Post-Quantum Cryptography - PQCrypto 2018*, pages 292–311, 2018.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_48