# Extreme Enumeration on GPU and in Clouds
## - How Many Dollars You Need to Break SVP Challenges -

Po-Chun Kuo[1], Michael Schneider[2], Özgür Dagdelen[3], Jan Reichelt[3], Johannes Buchmann[2,3], Chen-Mou Cheng[1], and Bo-Yin Yang[4]

[1] National Taiwan University, Taipei, Taiwan
[2] Technische Universität Darmstadt, Germany
[3] Center for Advanced Security Research Darmstadt (CASED), Germany
[4] Academia Sinica, Taipei, Taiwan

**Abstract.** The complexity of the Shortest Vector Problem (SVP) in lattices is directly related to the security of NTRU and the provable level of security of many recently proposed lattice-based cryptosystems. We integrate several recent algorithmic improvements for solving SVP and take first place at dimension 120 in the SVP Challenge Hall of Fame. Our implementation allows us to find a short vector at dimension 114 using 8 NVIDIA video cards in less than two days.

Specifically, our improvements to the recent Extreme Pruning in enumeration approach, proposed by Gama *et al.* in Eurocrypt 2010, include: (1) a more flexible bounding function in polynomial form; (2) code to take advantage of Clouds of commodity PCs (via the MapReduce framework); and (3) the use of NVIDIA's Graphics Processing Units (GPUs). We may now reasonably estimate the cost of a wide range of SVP instances in U.S. dollars, as rent paid to cloud-computing service providers, which is arguably a simpler and more practical measure of complexity.

**Keywords:** Shortest Vector Problem, GPU, Cloud Computing, Enumeration, Extreme Pruning.

## 1 Introduction

Lattice-based cryptography is a hot topic, with numerous submissions and publications at prestigious conferences in the last two years. The reasons that it might have become so popular include:

- lattice-based PKCs, unlike ECC, do not immediately succumb to large quantum computers (i.e., they are "post-quantum");
- lattice-based PKCs enjoy the (so far) unique property of being protected by a worst-case hardness assumption (i.e., they are unbreakable if *any* of a large class of lattice-based problem *at a lower dimension* is intractable);
- lattices can be used to create fully homomorphic encryptions.

One of the main problems in lattice-based cryptography is the *shortest vector problem* (SVP). As the name implies, it is a search for a non-zero vector with the smallest Euclidean norm in a lattice. The SVP is NP-hard under randomized reductions. The *approximate shortest vector problem* (ASVP) is the search for a short non-zero vector whose length is at most some given multiple of the minimum. It is easy in some cases, as shown by the LLL algorithm [LLL82]. Although LLL has polynomial running time, the approximation factor of LLL is exponential in the lattice dimension. The complexity of SVP (and ASVP) has been studied for decades, but practical implementations that take advantage of special hardware are not investigated seriously until recently [HSB$^+$10, DS10, DHPS10].

In contrast, enumeration is another way to solve SVP and ASVP, which can be viewed as a depth-first search in a tree structure, going over all vectors in a specified search region deterministically. Typically, a basis transformation such as BKZ [SE94] is performed first to improve the basis to one likely to yield a short vector via enumeration.

At Eurocrypt 2010, Gama *et al.* proposed the *Extreme Pruning* approach to solving SVP and ASVP [GNR10] and showed that it is possible to speed up the enumeration exponentially by randomizing the algorithm. The idea is that, instead of spending a lot of time searching *one* tree, one generates *many* trees and only spends a small amount of time on each of them by aggressively pruning the subtrees unlikely to yield short vectors using a bounding function. That is, one focuses on the parts of the trees that are more "fruitful" in terms of the likelihood of producing short vectors per unit time spent.

In other words, one should try to maximize the success probability of finding a short vector *per unit of computing time spent* by choosing an appropriate bounding function in pruning. Therefore, which bounding function works better depends on the particular *implementation.*

In this paper, we make a practical contribution on several fronts.

1. We integrate the Extreme Pruning idea of Gama *et al.* [GNR10] into the GPU implementation of [HSB$^+$10].
2. We extend the implementation by using multiple GPUs and run it on Amazon's EC2 in order to harness the immense computational power of such cloud services.
3. We extrapolate our average-case run times to estimate the run time of our implementation for solving ASVP instances of the SVP Challenge in higher dimensions.
4. Consequently, we set new records for the SVP challenge in dimensions 114, 116, and 120. The previous record was for dimension 112.

As a result, the average "cost" of solving ASVP (and breaking lattice-based cryptosystems) with our implementation can henceforth be measured directly in U.S. dollars, taking Lenstra's *dollarday* metric [Len05] to a next level[1]. That is, the cost will be shown literally as an amount on your invoice, e.g., the effort in our solving a 120-dimensional instance of the SVP Challenge translates to a 2300

[1] Before the final version went to press, it is brought to our attention that, unbeknownst to us, Kleinjung, Lenstra, Page, and Smart had also started to adopt a similar metric in an ePrint report [KLPS11] dated May 2011.

USD bill from Amazon. Moreover, this new metric is more practical in that the parallelizability of the algorithm or the parallelization of the implementation is *explicitly* taken into account, as opposed to being assumed or unspecified in the dollarday metric. Needless to say, such a cost should be understood as an upper bound obtained based on our implementation, which can certainly be improved, e.g., by using a better bounding function or better programming.

## 2 Preliminaries

### 2.1 Lattices, Algorithms, and SVP

Let $m, n \in \mathbb{Z}$ with $n \leq m$, and let $\mathbf{b}_i \in \mathbb{Z}^m$ for $1 \leq i \leq n$ be a set of linearly independent vectors. The set of all integer linear combinations of the vectors $\mathbf{b}_i$ is called a lattice $\Lambda$: $\Lambda = \{\sum_{i=1}^{n} x_i \mathbf{b}_i \mid x_i \in \mathbb{Z}\}$ . The matrix $\mathbf{B} \in \mathbb{Z}^{m \times n}$ consisting of the column vectors $\mathbf{b}_i$ is called a basis of $\Lambda$, we write $\Lambda = \Lambda(\mathbf{B})$. $\Lambda$ is an additive group in $\mathbb{Z}^m$. If $n = m$ the lattice is called full-dimensional. The basis of a lattice is not unique. The product of a basis with an unimodular matrix $\mathbf{M}$ ($|\det(\mathbf{M})| = 1$) does not change the lattice. The value $\lambda_1(\Lambda(\mathbf{B}))$ denotes the norm of a shortest non-zero vector in the lattice. It is called the first minimum. The determinant of a lattice is the value $\det(\Lambda(\mathbf{B})) = \sqrt{\det(\mathbf{B}^t\mathbf{B})}$. If $\Lambda(\mathbf{B})$ is full-dimensional, then the lattice determinant is equal to the absolute value of the determinant of the basis matrix ($\det(\Lambda(\mathbf{B})) = |\det(\mathbf{B})|$). In the remainder of this paper, we will only be concerned with full-dimensional lattices. The determinant of a lattice is independent of the basis; if the basis changes, the determinant remains the same.

The shortest vector problem in lattices is stated as follows. Given a lattice basis $\mathbf{B}$, output a vector $\mathbf{v} \in \Lambda(\mathbf{B}) \setminus \{0\}$ subject to $\|\mathbf{v}\| = \lambda_1(\Lambda(\mathbf{B}))$. The problem that we address in the remainder of this paper is the following: given a lattice basis $\mathbf{B}$ and a norm bound $A$, find a non-zero vector $\mathbf{v} \in \Lambda(\mathbf{B})$ subject to $\|\mathbf{v}\| \leq A$.

The Gaussian heuristic assumes that the number of lattice points inside a set $S$ is approximately $\mathrm{vol}(S)/\det(\Lambda)$. Using this heuristic and the volume of a unit sphere in dimension $n$, we can compute an approximation of the first minimum of the lattice $\Lambda$: $FM(\Lambda) = \frac{\Gamma(n/2+1)^{1/n}}{\sqrt{\pi}} \cdot \det(\Lambda)^{1/n}$ . Here $\Gamma(x)$ is the gamma-function. This estimate is used, among others, to predict the length of shortest vectors in the SVP challenge [GS10]. In our experiments as well as in the SVP challenge the heuristic shows to be a good estimate of a shortest vector length for the lattices used. Throughout the rest of this paper, our goal will always be to find a vector below $1.05 \cdot FM(\Lambda)$, the same as in the SVP challenge.

The Gram-Schmidt orthogonalization (GSO) of a matrix $\mathbf{B} \in \mathbb{Z}^{n \times n}$ is $\mathbf{B}^* = [\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*] \in \mathbb{R}^{n \times n}$. It is computed via $\mathbf{b}_i^* = \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{i,j} \mathbf{b}_j^*$ for $i = 1, \ldots, n$, where $\mu_{i,j} = \mathbf{b}_i^T \mathbf{b}_j^* / \|\mathbf{b}_j^*\|^2$ for all $1 \leq j \leq i \leq n$. We have $\mathbf{B} = \mathbf{B}^* \mu^T$, where $\mathbf{B}^*$ is orthogonal and $\mu^T$ is an upper triangular matrix. Note that $\mathbf{B}^*$ is not necessarily a lattice basis. The values $\mu$ are called the Gram-Schmidt coefficients.

The LLL [LLL82] and the BKZ [SE94] algorithms can be used for pre-reduction of lattices, before running an SVP algorithm. Pre-reduction speeds up the enumeration, since the size of the enumeration tree is depending on the quality of the input basis. BKZ is controlled by a blocksize parameter $\beta$, and LLL is the special case of BKZ with parameter $\beta = 2$. Higher blocksize guarantees a better reduction quality, in the sense that vectors in the basis are shorter and the angles between basis vectors are closer to orthogonal. The gain in reduction quality comes at the cost of increasing runtime. The runtime of BKZ increases exponentially with the blocksize $\beta$. In the lattice dimension, the runtime of BKZ behaves polynomial in practice, whereas no proof of this runtime is known. The overall runtime of our SVP solver will include the BKZ pre-reduction run times as well as enumeration run times. It is an important issue to find suitable blocksize parameters for pre-reduction.

**Algorithms for SVP.** There are mainly three different approaches how to solve the shortest vector problem. First, there are probabilistic sieving algorithms [AKS01, NV08, MV10b]. They output a solution to SVP with high probability only, but allow for single exponential runtime. The most promising sieving candidate in practice at this time is the GaussSieve algorithm [MV10b]. Further, there exists an algorithm based on Voronoi cell computation [MV10a]. This is the first deterministic SVP algorithm running in single exponential time, but experimental results lack so far. Third, there is the group of enumeration algorithms that perform an exhaustive search over all lattice points in a suitable search region. Based on the algorithms by Kannan [Kan83] and Fincke/Pohst [FP83], Schnorr and Euchner presented the ENUM algorithm [SE94]. It was analyzed in more details in [PS08]. The latest improvement called extreme pruning providing for huge exponential speedups, was shown by Gama, Nguyen, and Regev [GNR10]. In the remainder of this paper, we will only be concerned with extreme pruned enumeration, since this variant of enumeration is the strongest SVP solver at this time.

Ideas for parallel enumeration for shortest vectors were presented in [HSB$^+$10] for GPUs, in [DS10] for multicore CPUs, and in [DHPS10] for FPGAs. Concerning extreme pruning, there is no parallel version known to us to date, even no serial implementation is publicly available.

The lattices that we use for our tests throughout this paper are those of the SVP challenge [GS10]. They follow the ideas of the lattices from [GM03], and are used for testing SVP and lattice reduction algorithms, e.g., in [GN08, HSB$^+$10].

### 2.2 Enumeration and Extreme Pruning

Here we will present the basic idea of enumeration for shortest vectors. $n$ denotes the dimension of the full-dimensional lattices. To find a shortest non-zero vector of a lattice $\Lambda(\mathbf{B})$ with $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$, ENUM takes as input the Gram-Schmidt coefficients $(\mu_{i,j})_{1 \leq j \leq i \leq n}$, the quadratic norm of the Gram-Schmidt orthogonalization $\|\mathbf{b}_1^*\|^2, \ldots, \|\mathbf{b}_n^*\|^2$ of $\mathbf{B}$, and an initial search bound $A$.

The search space is the set of all coefficient vectors $\mathbf{u} \in \mathbb{Z}^n$ that satisfy $\|\sum_{t=1}^{n} u_t \mathbf{b}_t\| \leq A$. Starting with an LLL-reduced basis, it is common to set

$A = \|\mathbf{b}_1^*\|^2$ in the beginning. If the norm of the shortest vector is known beforehand, it is possible to start with a lower $A$, which limits the search space and reduces the runtime of the algorithm. In the equation

$$\left\|\sum_{t=1}^{n} u_t \mathbf{b}_t\right\| = \min_{x \in \mathbb{Z}^n} \left\|\sum_{t=1}^{n} x_t \mathbf{b}_t\right\|$$

we replace all $\mathbf{b}_t$ by their orthogonalization, i.e., $\mathbf{b}_t = \mathbf{b}_t^* + \sum_{j=1}^{t-1} \mu_{t,j} \mathbf{b}_j^*$ and get

$$\left\|\sum_{t=1}^{n} u_t \mathbf{b}_t\right\|^2 = \left\|\sum_{t=1}^{n} \left(u_t(\mathbf{b}_t^* + \sum_{j=1}^{t-1} \mu_{t,j} \mathbf{b}_j^*)\right)\right\|^2 = \sum_{t=1}^{n} (u_t + \sum_{i=t+1}^{n} \mu_{i,t} u_i)^2 \|\mathbf{b}_t^*\|^2 .$$

For index $k$, enumeration is supposed to check all coefficient vectors $\mathbf{u}$ with

$$\sum_{t=n+1-k}^{n} (u_t + \sum_{i=t+1}^{n} \mu_{i,t} u_i)^2 \cdot \|\mathbf{b}_t^*\|^2 < A \quad , \quad 1 \le k \le n . \tag{1}$$

For index $t$, the summand is independent of values with lower index $< t$. This means that changing the coefficient $u$ for lower indices $< t$ does not affect the upper part of the sum with index $\geq t$. Therefore, the indices are arranged in a tree structure, where the root node contains values for coefficient $u_n$, intermediate nodes contain partly filled coefficient vectors $(\times, u_t, \ldots, u_n)$, and leaf nodes contain full linear combinations $(u_1 \ldots u_n)$. Here the symbol $\times$ denotes that the first values of the coefficient vector are not set. Since the $\|b_i^*\|$ are orthogonal, the sum can only increase when we step a layer down in the tree, the sum will never decrease. Therefore, when an inner node of the tree has extended the search norm $A$, we can cut off the whole subtree rooted at this node and skip enumerating the subtree.

Schnorr and Hörner already presented an idea to prune some of the subtrees that are unlikely to contain a shorter vector [SH95]. Their pruned enumeration runs deterministically with a certain probability to miss a shortest vector. The [SH95] pruning idea was analyzed and improved in [GNR10][2]. Instead of using the same norm bound $A$ on every layer of the enumeration tree (Equation (1)), Gama et al. introduce a bounding vector $(R_1, \ldots, R_n) \in [0, 1]^n$, with $R_1 \leq \ldots \leq R_n$. $A$ on the right side of the testing condition (1) is replaced by $R_k \cdot A$. It can be shown that, assuming various heuristics [GNR10], the lattice vectors cut off by this approach only contain a shortest vector with low probability.

With this pruning technique, an exponential speedup compared to deterministic enumeration can be gained. In the original paper, various bounding function vectors were presented in theory. For the experiments, the authors use a numerically optimized function.

[2] The authors of [GNR10] also showed some flaws in the analysis of [SH95].

### 2.3 Cloud Computing, Amazon EC2, and GPU

Cloud computing is an emerging computing paradigm that allows data centers to provide large-scale computational and data-processing power to the users on a "pay-as-you-go" basis. Amazon Web Services (AWS) is one of the earliest and major cloud-computing providers, who provides, as the name suggests, web services platforms in the cloud. The Elastic Compute Cloud (EC2) provides compute capacity in the cloud as a foundation for the other products that AWS provides. With EC2, the users can rent large-scale computational power on demand in the form of "instances" of virtual machines of various sizes, which is charged on an hourly basis. The users can also use popular parallel computing paradigms such as the MapReduce framework [DG04], which is readily available as the AWS product "Elastic MapReduce." Furthermore, such a centralized approach also frees the users from the burden of provisioning, acquiring, deploying, and maintaining their own physical compute facilities.

Naturally, such a paradigm is economically very attractive for most users, who only need large-scale compute capacity occasionally. For large-scale computations, it may be advisable to buy machines instead of renting them because Amazon presumably expects to make a profit on renting out equipment, so our extrapolation might over-estimate the cost for long-term computations. However, we believe that these cloud-computing service providers will become more efficient in the years to come if cloud computing indeed becomes the mainstream paradigm of computing. Moreover, trade rumors has it that Amazon's profit margins are around 0% (break-even) as of mid-2011, and nowhere close to 100%, so we can say confidently that Amazon rent cannot be more than $2\times$ what a large-scale user would have spent if he bought and maintained his own computers and networking. Thus, Amazon prices can still be considered a realistic measure of computing cost and a good yardstick for determining the strength of cryptographic keys.

In estimating complexity such that of solving (A)SVP or problems of the same or similar nature, Amazon EC2 can be used to provide a common measure of cost as a metric in comparing alternative or competing cryptanalysis algorithms and their implementations. Moreover, when using the Amazon EC2 metric, the parallelizability of the algorithm or the parallelization of the implementation is *explicitly* taken into account, as opposed to being assumed or unspecified. In addition to its simplicity, we argue that the EC2 metric is more practical than the *dollardays* metric of [Len05], and a recent report by Kleinjung, Lenstra, Page, and Smart [KLPS11] also agrees with us in taking a similar approach and measure with Amazon's EC2 cloud.

Graphics processing units (GPUs) represent another class of many-core architectures that are cost-effective for achieving high arithmetic throughput. The success of GPU has mainly been driven by the economy of scale in the video game industry. Currently, the most widely used GPU development toolchain is NVIDIA's CUDA (Compute Unified Device Architecture) [KH10]. At the core of CUDA are three key abstractions, namely, a hierarchy of thread groups, shared memories, and barrier synchronization, that are exposed to the programmers as a set of extensions to the C programming language. At the system level, the GPU is used as a coprocessor to the host processor for massively data-parallel computations, each of which is executed by a grid of GPU threads that must run the same program (the kernel). This is the SPMD (single program, multiple data) programming model, similar to SIMD but with more flexibility such as in changing of data size on a per-kernel-launch basis, as well as deviation from SIMD to MIMD at a performance penalty.

AWS offers several different compute instances for their customers to choose based on their computational needs. The one that interests us the most is the largest instance called "Cluster Compute Quadruple Extra Large" (`cc1.4xlarge`) which is designed for high-performance computing. Each such instance consists of 23 GB memory provide 33.5 "EC2 Compute Units" where each unit roughly "provides the equivalent CPU capacity of a 1.0–1.2 GHz 2007 Opteron or 2007 Xeon processor," according to Amazon.

Starting from late 2009, AWS also adds to its inventory a set of instances equipped with GPUs, which is called "Cluster GPU Quadruple Extra Large" (`cg1.4xlarge`), which is basically a `cc1.4xlarge` plus two NVIDIA Tesla "Fermi" M2050 GPUs. As of the time of writing, the prices for renting the above compute resources are shown in Table 1. The computation time is always rounded up to the next full hour for pricing purposes.

**Table 1.** Pricing information from `http://aws.amazon.com/ec2/pricing/`

| | Elastic Compute Cloud | 1 Year Reserved Pricing | Elastic MapReduce |
|---|---|---|---|
| `cc1.4xlarge` | 1.60 USD/hour | 4290 USD + 0.56 USD/hour | 0.33 USD/hour |
| `cg1.4xlarge` | 2.10 USD/hour | 5630 USD + 0.74 USD/hour | 0.42 USD/hour |

For computations lasting less than 172 days it is cheaper to use on-demand pricing. For longer runs, there is an option to "reserve" an instance for 1 year (or even 3), which means that the user pays an up-front cost (see table above) to cut the on-demand cost of these instances.

## 3 Implementation

For each randomized basis, we use LLL-XD followed by BKZ-FP of the NTL Library [Sho] with $\delta = 0.99$, different blocksizes $\beta$, and pruning parameter $p = 15$. As already mentioned above, the problem we address is finding a vector below a search bound $1.05 \cdot FM$ that heuristically guesses the length of a shortest vector of the input lattice. Adapting our implementations to other goal values is straight forward. It will only change the success probability and the runtime, therefore, we have to fix the bound for this work.

### 3.1 Bounding Function

As mentioned above, selecting a suitable bounding function is an important part of extreme enumeration. It influences the runtime as well as the success probability of each enumeration tree. The bounding function we use is a polynomial

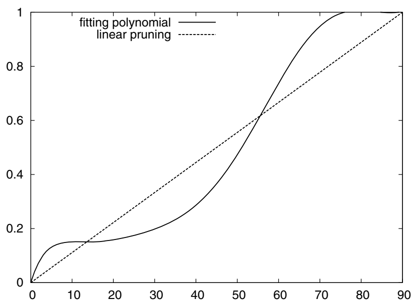


**Fig. 1.** Polynomial bounding function $p(x)$, scaled to lattice dimension 90. The dashed line shows a linear bounding function.

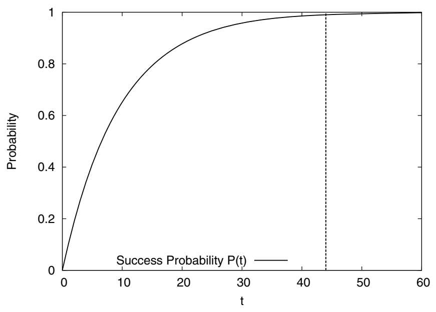


**Fig. 2.** Success probability of Extreme Enum assuming a success probability $p_{succ} = 10\%$ for one single tree. On average, we have to start 44 trees to finish with success probability $> 99\%$

$p(x)$ of degree eight that aims to fit the best bounding function of [GNR10] in dimension 110. We use

$$p(x) = \sum_{i=0}^{8} v_i x^i$$

where $\mathbf{v} = (9.1 \cdot 10^{-4}, 4 \cdot 10^{-2}, -4 \cdot 10^{-3}, 2.3 \cdot 10^{-4}, -6.9 \cdot 10^{-6}, 1.21 \cdot 10^{-7}, -1.2 \cdot 10^{-9}, 6.2 \cdot 10^{-12}, -1.29 \cdot 10^{-14})$ to fit the 110-dimensional bounding function. For dimension $n$ we use $p(x \cdot 110/n)$. Figure 1 shows our polynomial bounding function $p(x)$, scaled to dimension 90.

Using an MPI-implementation for CPU we gained a success probability of finding a vector below $1.05 \cdot FM(\Lambda)$ of $p_{succ} > 10\%$. We use 10 lattice bases in each dimension and run BKZ and enumeration on up to 1000 randomized instances for each basis. We stop each lattice after 5 hours of computation, so that the total time is still manageable. In dimensions 96 we increase the maximum time from 5 to 20 hours. In total, we have up to 1000 trees in each dimension to compute the success probability of our bounding function. For a comparable bounding function, the authors of [GNR10] get a much smaller success probability. This is due to the fact that in our case we expect about $1.05^n$ many vectors below the larger search bound, whereas the analysis of Gama et al. assumes that only a single vector exists below their bound.

Figure 9 in Appendix A shows the expectation values of the success of BKZ and ENUM. More exactly, it shows the expectation value $E(X)$ of $P(X \leq t)$, which gives a success probability of $p = 1/E(X)$. For higher dimensions $m > 90$ the success probability of BKZ tends to zero in every tested case. $P(t) = 1 - (1 - p_{succ})^t$ is the success probability to find a shortest vector below $1.05 \cdot FM(\Lambda)$ when starting $t$ enumeration trees in parallel. Figure 2 shows the success probability $P$ for $p_{succ} = 10\%$. This implies that on average we have to start 44

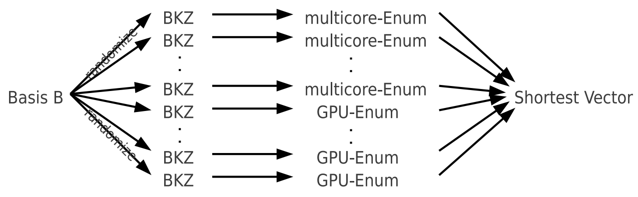


**Fig. 3.** The model of our parallel SVP solver. The basis **B** is randomized, and each instance is solved either on CPU or on GPU. In the end, the shortest of all found vectors is chosen as output. Since we use pruned enumeration, not all instances will find a vector below the given bound.

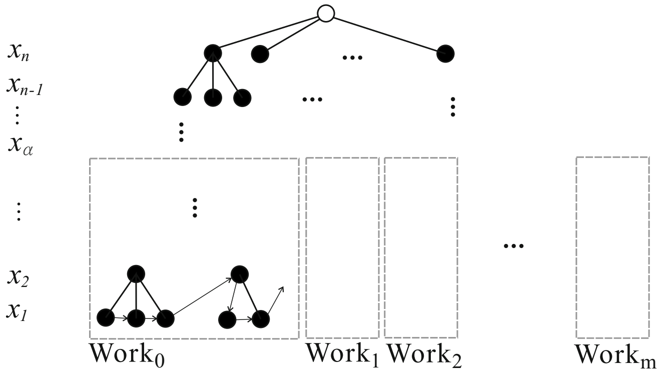


**Fig. 4.** Illustration of the parallel enumeration process. The top tree $x_n, x_{n-1}, ..., x_\alpha$ is enumerated on a single core, and the lower trees $x_{\alpha-1}, ..., x_2, x_1$ are explored in parallel on many mappers.

trees to find a vector below the given bound with probability $P(t) > 99\%$ (and not $1/p_{succ}$ many trees, as one could imagine).

## 3.2 Parallelization of Extreme Pruning Using GPU and Clouds

Our overall parallelization strategy follows the model shown in Figure 3. For success, it is sufficient if one randomized instance of ENUM finishes. The number of instances we start depends on the success probability of each instance, which itself is depending on the bounding function used. The high-level algorithm run by each multicore-Enum or GPU-Enum instance is illustrated in Figure 4.

For the calculation of the cost, it makes no difference if we use 8 cores for a multicore-tree or only one core. In practice, however, we can stop the whole computation if one of the trees has found a vector below the bound. Therefore, using multiple cores for enumeration may have some influence on the running time.

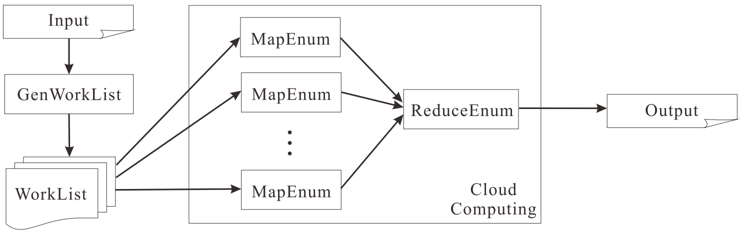


**Fig. 5.** Illustration of our MapReduce implementation of the enumeration algorithm

**GPU Implementation.** We used the implementation of [HSB+10] and included pruning according to [GNR10]. The GPU enumeration uses enumeration on top of the tree, which is performed on CPU, to collect a huge number of *starting points*, as shown in Figure 4. These starting points are vectors $(\times, \dots, \times, x_{n-\alpha+1}, \dots, x_n)$, where only the last $\alpha$ coefficients are set. A starting point can be seen as the root of a subtree in the enumeration tree. All starting points are copied to the GPU and enumerated in parallel. Due to load balancing reasons, this approach is done iteratively, until no more start points exist on top of the tree (see [HSB+10] for more details).

Since the code of extreme pruning only changes a few lines compared to usual enumeration, including pruning to the GPU implementation is straight forward. The improvement mentioned in [GNR10] concerning storage of intermediate sums was in parts already contained in the [HSB+10] implementation, so only slight changes were integrated into the GPU ENUM.

The GPU implementation allows the usage of different bounding functions, but for simplicity reasons we stick to the polynomial function specified above. Our implementation is available online[3].

**MapReduce Implementation.** Our MapReduce implementation is also based on [HSB+10]. The overall search process is illustrated in Figure 5. Specifically, we divide the search tree to top and lower trees. A top tree, which consists of levels $x_n$ through $x_\alpha$, is enumerated by a single thread in a DFS fashion, outputting all possible starting points $(x_\alpha, \dots, x_n)$ to a WorkList. When a mapper receives a starting point $(x_\alpha, \dots, x_n)$ from the WorkList, it first populates the unspecified coordinates $x_1, \dots, x_{\alpha-1}$ and obtains the full starting point

$$(x_1 = \lceil -\sum_{k=2}^{n} \mu_{k,1}, x_k \rfloor, \dots, x_{\alpha-1} = \lceil -\sum_{k=\alpha}^{n} \mu_{k,\alpha-1}, x_k \rfloor, x_\alpha, ..., x_n).$$

It then starts enumerating the lower tree from level 1 through $\alpha - 1$.

Because we scan the coefficients in a zigzag path, the lengths of the starting points usually show an increasing trend from the first to the last starting point. This can result uneven work distribution among the mappers. Therefore, we

[3] http://homes.esat.kuleuven.be/~jhermans/gpuenum/index.html

subdivide and randomly shuffle the WorkList so that each mapper gets many random starting points and hence have roughly equal amount of work among themselves. The effect is evident from the fact that the load-balancing factor, i.e., average running time divided by that of the slowest mapper, increases from 24% to 90%.

## 4 Experimental Results

In this section, we present the experimental results for our algorithmic improvements and parallel implementations on GPU and with MapReduce.

### 4.1 GPU Implementation

The GPU enumeration using extreme pruning solved the 114-dimensional SVP-challenge in about 40 hours using one single workstation with eight NVIDIA GeForce GTX 480 cards in parallel. Each GTX 480 has one GPU with 480 cores running at 1.4 GHz. The performance decreases from 200 Msteps/s to $\approx$ 100 Msteps/s using polynomial bounding function compared with an instance without pruning. With linear pruning, the decrease is less noticeable, but still apparent. This decrease is caused by the fact that subtrees are much thinner when pruning the tree. The number of starting points per second increases a lot, which coincides with the fact that subtrees, even though their dimension is much bigger now, are processed faster than without pruning.

We use 10 different lattices of the SVP challenge in each dimension 80–104 on the workstation equipped with eight GTX 480 cards to generate the timings of Figures 6 and 7.

*Workload Distribution between BKZ and ENUM.* We note that in general, if we spend more time in BKZ to produce a better basis, we would have a higher probability of finding a short vector in the subsequent ENUM phase. A natural question is, what is the optimal breakdown of workload between BKZ and ENUM?

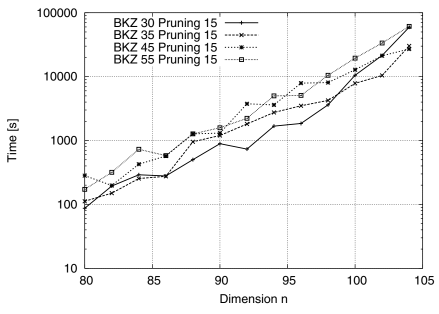


**Fig. 6.** Total running time for solving SVP instances from dimension 80 to 104

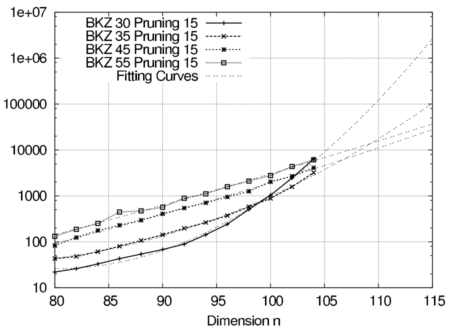


**Fig. 7.** Running time for one round of pruned ENUM, including fitting curves $t_{30}(n)$ to $t_{55}(n)$

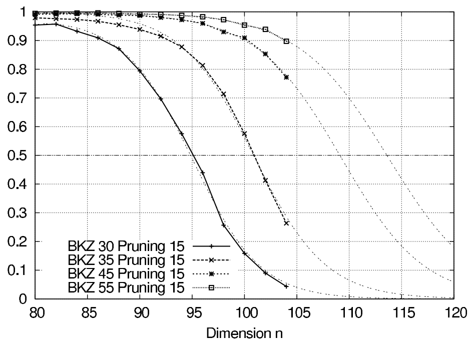


**Fig. 8.** Ratio of BKZ runtime to total runtime for a single enumeration tree

We conjecture that the distribution should be roughly equal, as is supported by empirical evidence that we obtained from our experiments (cf. Figure 8). In our experiments, BKZ 40 performs the best in 104-dimensional instances, whereas in Figure 8, it has a ratio that is the closest to 0.5. Similar trends can be observed for dimensions 86–97, for which the best BKZ block size is 30.

We use the data shown in Figure 8 to assess which of the curves from Figure 7 is the fastest one, and we use the extrapolation of this curve gained from data in dimension 80–104. This results in the cost function shown in Conjecture 1.

**Conjecture 1 (GPU timing function).** *Running BKZ and our implementation of pruned enumeration once on an NVIDIA GTX-480 GPU takes time*

$$time_{GPU}(n) = \begin{cases} t_{30}(n) = 2^{0.0138n^2-2.2n+93.2} & \textit{for } n \leq 97 \\ t_{35}(n) = 2^{0.0064n^2-0.92n+38.4} & \textit{for } 98 \leq n \leq 104 \\ t_{45}(n) = 2^{0.001n^2+0.034n-2.8} & \textit{for } 105 \leq n \leq 111 \\ t_{55}(n) = 2^{0.00059n^2+0.11n-5.8} & \textit{for } 112 \leq n \end{cases} \quad sec.$$

A more theoretic way to extrapolate the runtime would be to compute BKZ reduced bases, note the slope of the orthogonalized basis vectors, and use the runtime function of [GNR10] to compute the runtime. This approach ignores the runtime of BKZ (which is up to 50%) of the total runtime and relies on the Gaussian heuristic, while we are interested in practical runtime.

From the regression results shown in Figures 6 and 7, we can see that the run times for BKZ and ENUM are indeed polynomial and super-exponential, respectively. However, we notice that a larger BKZ block size does have a positive effect on the per-round running time of subsequent ENUM.

One difference is that Amazon uses M2050 GPU, not GTX480 (like in our experiments). The M2050 has better double precision performance. Since many operations in enumeration are performed using double precision operations, we expected a huge speed-up for enumeration. However, tests on M2050 GPUs did not show large speed-ups. One possible explanation is as follows. On the GPU, many additional operations have to be performed in integer-precision in order to split the work and reach a good load balancing. Therefore, double-precision operations are less than a fourth of the total number of operations, which makes the speed-ups on M2050 GPUs minor.

### 4.2 MapReduce Implementation

Our MapReduce implementation is compiled by `g++` version 4.4.4 x86_64 with the options `-O9 -ffast-math -funroll-loops -ftree-vectorize`. Using the MapReduce implementation, we are able to solve the 112-dimensional SVP-challenge in a few days. More exactly, we were using 10 nodes, 84 physical cores (totaling 140 virtual cores as some of the cores are hyperthreaded), which gives a total number of 334 GHz.

We note that the bounding function used in this computation is different from the polynomial bounding function described earlier. We were lucky in that only after 101 hours, or 1/9 of the estimated time, a shorter vector was found. We also noticed that the runtime scales linearly with the number of CPU cores used in total, meaning if we increase the number of CPU cores by a factor of 10, the runtime will decrease by factor 10.

Overall, from the test data of solving SVPs at dimension 100, 102, and 104 using the same set of seeds, we found that a GTX480 is roughly two to three times faster than a four-core, 2.4 GHz Intel Core i7 processor for running our SVP solvers. We conjecture that the running time for our MapReduce implementation is also similar to that of our GPU implementation, as shown in Conjecture 1.

### 4.3 Final Pricing

We use Conjecture 1 to derive the final cost function for solving SVP challenges in higher dimensions $n \geq 112$. Recall that Amazon instances have to be paid for complete hours, therefore we round the runtime in hours to the next highest integer value. Using 44 enumeration trees leads to a success probability of at least 99%.

**Conjecture 2 (Final Pricing).** *Solving an SVP challenge with our implementation in dimension* $n \geq 112$ *with a success probability of* $\geq 99\%$ *on Amazon EC2 (using on demand pricing) costs*

$$cost_{GPU}(n) = \lceil time_{GPU}(n)/3600 \rceil \cdot 44 \cdot 2.52 \ \mathit{USD}.$$

Following Conjecture 2 solving the 120-dimensional instance of the challenge costs 1,885 USD, which is a bit less than the amount we paid for practically solving it (due to conservative reservation of compute resources on EC2). We actually fired up 50 cg1.4xlarge instances for a total of 946 instance-hours, and incurred a bill of 2,300 USD. For instance, solving the 140-dimensional challenge would cost roughly 72,405 USD.

## 5 Concluding Remarks and Further Work

*Cryptographic Key Sizes.* The ability of solving SVP does not directly affect cryptographic schemes based on lattice problems. The hardness of lattice-based signature schemes is mostly based on the SIS problem, whereas the hardness of encryption schemes is mostly based on the LWE problem. Both the SIS and the LWE problem can be proven to be as hard as the SVP in lattices of a smaller dimension (so-called *worst-case to average-case reduction*). That means that a successful attacker of a cryptographic system is able to solve SVP in all lattices of a smaller dimension. This implies that our cost estimates for SVP can be used to assess the hardness of the basic problem of cryptosystems only.

Real attacks on cryptosystems mostly apply approximation algorithms, like BKZ. Since enumeration can be used as a subroutine there, speeding up enumeration also affects direct attacks on lattice based cryptosystems.

*Further Work.* For GPUs the need of finding new bounding functions seems apparent. Since trees are very thin when our polynomial bounding is applied the performance of the GPU decreases. Finding a new bounding function that allows for the same success probability but guarantees better performance will show the strength of the GPU even more. Besides that, it is an open problem which bounding function gives the best performance in practice, be it on CPU or GPU.

**Acknowledgements.** Schneider is supported by project BU 630/23-1 of the German Research Foundation (DFG). Dagdelen was supported by CASED (`www.cased.de`). Cheng, Kuo and Yang are supported by National Science Council, National Taiwan University and Intel Corporation under Grants NSC99-2911-I-002-201, 99R70600, and 10R70500, and Yang also by Academia Sinica Career Award. We thank Paul Baecher and Pierre-Louis Cayrel for their comments on an earlier version of this paper, Phong Nguyen and Oded Regev for their hints regarding the success probability of the bounding function, Pierre-Alain Fouque for his advice, and the anonymous reviewers for their helpful comments.

## References

[AKS01] Ajtai, M., Kumar, R., Sivakumar, D.: A sieve algorithm for the shortest lattice vector problem. In: STOC 2001, pp. 601–610. ACM, New York (2001)

[DG04] Dean, J., Ghemawat, S.: MapReduce: Simplified data processing on large clusters. In: OSDI 2004: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, USA (December 2004)

[DHPS10] Detrey, J., Hanrot, G., Pujol, X., Stehlé, D.: Accelerating Lattice Reduction with FPGAs. In: Abdalla, M., Barreto, P.S.L.M. (eds.) LATINCRYPT 2010. LNCS, vol. 6212, pp. 124–143. Springer, Heidelberg (2010)

[DS10] Dagdelen, Ö., Schneider, M.: Parallel Enumeration of Shortest Lattice Vectors. In: D'Ambra, P., Guarracino, M., Talia, D. (eds.) Euro-Par 2010. LNCS, vol. 6272, pp. 211–222. Springer, Heidelberg (2010)

[FP83] Fincke, U., Pohst, M.: Michael Pohst. A procedure for determining algebraic integers of given norm. In: van Hulzen, J.A. (ed.) ISSAC 1983 and EUROCAL 1983. LNCS, vol. 162, pp. 194–202. Springer, Heidelberg (1983)

[GM03] Goldstein, D., Mayer, A.: On the equidistribution of Hecke points. Forum Mathematicum 15(2), 165–189 (2003)

[GN08] Gama, N., Nguyen, P.Q.: Predicting lattice reduction. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 31–51. Springer, Heidelberg (2008)

[GNR10] Gama, N., Nguyen, P.Q., Regev, O.: Lattice Enumeration Using Extreme Pruning. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 257–278. Springer, Heidelberg (2010)

[GS10] Gama, N., Schneider, M.: SVP Challenge (2010), `http://www.latticechallenge.org/svp-challenge`

[HSB+10] Hermans, J., Schneider, M., Buchmann, J., Vercauteren, F., Preneel, B.: Parallel Shortest Lattice Vector Enumeration on Graphics Cards. In: Bernstein, D.J., Lange, T. (eds.) AFRICACRYPT 2010. LNCS, vol. 6055, pp. 52–68. Springer, Heidelberg (2010)

[Kan83] Kannan, R.: Improved algorithms for integer programming and related lattice problems. In: STOC 1983, pp. 193–206. ACM, New York (1983)

[KH10] Kirk, D.B., Hwu, W.-m.: Programming Massively Parallel Processors: A Hands-on Approach, 1st edn. Morgan Kaufmann, San Francisco (2010)

[KLPS11] Kleinjung, T., Lenstra, A.K., Page, D., Smart, N.P.: Using the cloud to determine key strengths. Cryptology ePrint Archive, Report 2011/254 (2011), `http://eprint.iacr.org/`

[Len05] Lenstra, A.: Key lengths. In: Bidgoli, H. (ed.) Handbook of Information Security. Wiley, Chichester (2005)

[LLL82] Lenstra, A., Lenstra, H., Lovász, L.: Factoring polynomials with rational coefficients. Mathematische Annalen 4, 515–534 (1982)

[MV10a] Micciancio, D., Voulgaris, P.: A deterministic single exponential time algorithm for most lattice problems based on voronoi cell computations. In: STOC, pp. 351–358. ACM, New York (2010)

[MV10b] Micciancio, D., Voulgaris, P.: Faster exponential time algorithms for the shortest vector problem. In: SODA 2010, pp. 1468–1480. ACM/SIAM (2010)

[NV08] Nguyen, P.Q., Vidick, T.: Sieve algorithms for the shortest vector problem are practical. J. of Mathematical Cryptology 2(2) (2008)

[PS08] Pujol, X., Stehlé, D.: Rigorous and Efficient Short Lattice Vectors Enumeration. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 390–405. Springer, Heidelberg (2008)

[SE94] Schnorr, C.-P., Euchner, M.: Lattice basis reduction: Improved practical algorithms and solving subset sum problems. Mathematical Programming 66, 181–199 (1994)

[SH95] Schnorr, C.-P., Hörner, H.H.: Attacking the chor-rivest cryptosystem by improved lattice reduction. In: Guillou, L.C., Quisquater, J.-J. (eds.) EUROCRYPT 1995. LNCS, vol. 921, pp. 1–12. Springer, Heidelberg (1995)

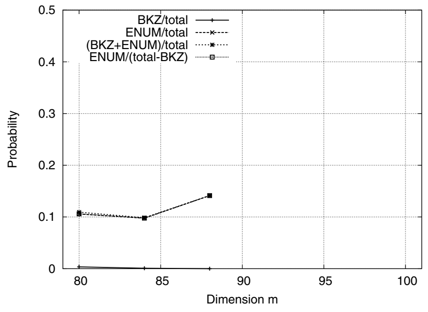
[Sho] Shoup, V.: Number theory library (NTL) for C++, version 5.5.2, `http://www.shoup.net/ntl/`

## A Success Probability Using $p(x)$

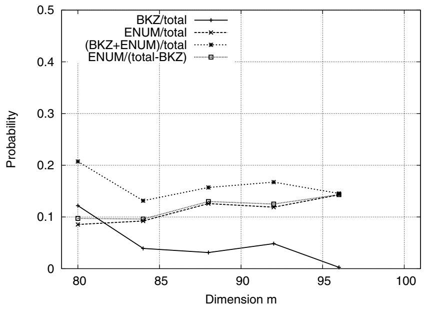
We ran experiments using the SVP challenge lattices, in order to assess the practical success probability (the probability of a single ENUM run to find a short vector) of extreme pruning using the polynomial bounding function $p(x)$. Using a multicore CPU implementation we started extreme pruning on up to $10,000$ lattices in each dimension (we stopped each experiment after 20 hours of computation). Figure 9 shows the average success rate of BKZ (with pruning parameter 15) and ENUM in dimensions 80 to 96 for different BKZ blocksizes. The values shown are the number of successfully reduced lattices divided by the number of started lattices in each dimension.

With BKZ blocksize 20, the pre-reduction was not strong enough, so neither BKZ nor ENUM could find a vector below the search bound in dimensions $\geq 96$ within 20 hours. In dimension 100, the number of finished enumeration trees was already too small to derive a meaningful success rate.

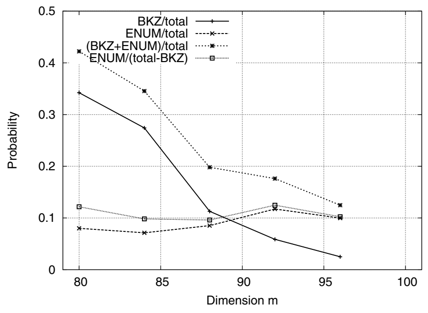
The success rate of BKZ vanishes in higher dimensions. For each BKZ blocksize, the success rate of ENUM stabilizes at a value $> 10\%$. Since the success rate is higher than this value in almost every case, we assume a value of $p_{succ} = 10\%$ for our polynomial bounding function $p(x)$.



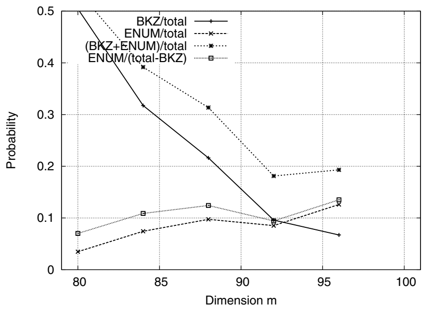
(a) BKZ-20



(b) BKZ-30



(c) BKZ-40



(d) BKZ-50

**Fig. 9.** Average values of success of the polynomial bounding function. total = number of samples; BKZ = number of samples solved by BKZ; ENUM = number of samples solved by pruned enumeration.