

Shortest Lattice Vector Enumeration on Graphics Cards ^{*}

Jens Hermans ^{**1}, Michael Schneider², Johannes Buchmann², Frederik Vercauteren ^{***1}, and Bart Preneel¹

¹ Katholieke Universiteit Leuven
{Jens.Hermans,Frederik.Vercauteren,Bart.Preneel}@esat.kuleuven.be

² Technische Universität Darmstadt
{mischnei,buchmann}@cdc.informatik.tu-darmstadt.de

Abstract. In this paper we make a first feasibility analysis for implementing lattice reduction algorithms on GPU using CUDA, a programming framework for NVIDIA graphics cards. The enumeration phase of the BKZ lattice reduction algorithm is chosen as a good candidate for massive parallelization on GPU. Given the nature of the problem we gain large speedups compared to previous CPU implementations. Our implementation saves more than 50% of the time in high lattice dimensions. Among other impacts, this result influences the security of lattice based cryptosystems.

Keywords: Lattice reduction, ENUM, parallelization, graphics cards, CUDA

1 Introduction

Lattice-based cryptosystems are considered to be secure against quantum computer attacks. Therefore those systems are promising alternatives to factoring or discrete logarithm based systems. For those systems already exist quantum computer algorithms that solve the problems efficiently. This will turn out as a problem as soon as large scale quantum computers will be built.

The security of lattice-based schemes is based on the hardness of special lattice problems. Lattice basis reduction helps to determine the actual hardness of those problems in practice. During the last ten years there have been no notable improvements to practical lattice reduction. So people start thinking about using special hardware to speed up the existing algorithms.

^{*} The work described in this report has in part been supported by the Commission of the European Communities through the ICT program under contract ICT-2007-216676. The information in this document is provided as is, and no warranty is given or implied that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

^{**} Research assistant, sponsored by the Fund for Scientific Research - Flanders (FWO).

^{***} Postdoctoral Fellow of the Fund for Scientific Research - Flanders (FWO).

In 2008, Bernstein et al. use parallelization techniques on graphic cards to solve integer factorization using elliptic curves [BCC⁺09]. Using NVidia’s CUDA parallelization framework, they gained a speed-up of up to 6 compared to computation on a four core CPU. Former applications of GPU parallelization in cryptography were mainly to secret key cryptography, see [CIKL05] for example.

Our Contribution. In this paper we present a parallel version of the enumeration algorithm that searches short vectors in a lattice. We are using the CUDA framework of NVIDIA for implementing an algorithm on graphics cards. Firstly we explain the ideas of how to parallelize enumeration on GPU. Secondly we present some first experimental results. Using the GPU, we reduce the time required for enumeration of a lattice in dimensions bigger than 50 to less than 50%. This result influences the security of lattice based cryptosystems, that is mostly based on the hardness of finding short vectors in a lattice.

The original algorithm for exhaustive search in lattices was presented by Fincke and Pohst [FP83] and Kannan [Kan83]. The algorithm used in practice today is the variant of Schnorr and Euchner [SE91]. In [PS08] Pujol and Stehlé analyze the stability of the enumeration when using floating point arithmetic. A parallel version of the enumeration is not known to date.

2 Preliminaries

A lattice is a discrete subgroup of \mathbb{R}^d . It can be represented by a basis matrix $\mathbf{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_n\}$. We call $L(\mathbf{B}) = \{\sum_{i=1}^n x_i \mathbf{b}_i, x_i \in \mathbb{Z}\}$ the lattice spanned by the basis vectors $\mathbf{b}_i \in \mathbb{R}^d$. The dimension n of a lattice is the number of linear independent vectors in the lattice, i.e. the number of basis vectors. When $n = d$ the lattice is called full dimensional.

The basis of a lattice is not unique. Every unimodular transformation \mathbf{M} , i.e. transformation with $\det \mathbf{M} = \pm 1$, turns a basis matrix \mathbf{B} into a second basis \mathbf{MB} of the same lattice.

The determinant of a lattice is defined as $\det(L(\mathbf{B})) = \sqrt{\det(\mathbf{B}^T \mathbf{B})}$. For full dimensional lattices we have $\det(L(\mathbf{B})) = |\det(\mathbf{B})|$. The determinant of a lattice is invariant of the choice of the lattice basis, which follows from the multiplicative property of the determinant and the fact that basis transformations have determinant 1.

The length of the shortest vector of a lattice $L(\mathbf{B})$ is denoted $\lambda_1(L(\mathbf{B}))$ or in short λ_1 if the lattice is uniquely determined.

The Gram-Schmidt orthogonalization computes an orthogonal projection of a basis. It is an efficient algorithm that outputs $\mathbf{B}^* = [\mathbf{b}_1^*, \dots, \mathbf{b}_n^*]$ and $\mu_{i,j}$ such that $\mathbf{B} = \mathbf{B}^* \cdot [\mu_{i,j}]$, where $[\mu_{i,j}]$ is an upper triangular matrix consisting of the Gram-Schmidt coefficients $\mu_{i,j}$ for $1 \leq j \leq i \leq n$. The orthogonalized matrix \mathbf{B}^* is not necessarily a basis of the lattice.

2.1 Lattice Basis Reduction

Problems. Some lattice bases are more useful than others. The goal of lattice basis reduction (or in short lattice reduction) is to find a basis consisting of short and almost orthogonal lattice vectors. More exactly, we can define some (hard) problems on lattices. The most important one is the *shortest vector problem* (SVP), which is looking for a vector $\mathbf{v} \in L \setminus \{\mathbf{0}\}$ with $\|\mathbf{v}\| = \lambda_1(L(\mathbf{B}))$. In most cases, the Euclidean norm $\|\cdot\|_2$ is considered. As the SVP is \mathcal{NP} -hard (at least under randomized reductions) people consider the approximate version γ -SVP, that is looking for a vector $\mathbf{v} \in L \setminus \{\mathbf{0}\}$ with $\|\mathbf{v}\| \leq \gamma \cdot \lambda_1(L(\mathbf{B}))$.

Other important problems like the *closest vector problem* (CVP) that searches for a nearest lattice vector to a given point in space, its approximation variant γ -CVP, or the *shortest basis problem* (SBP) are listed and described in detail in [ECR].

Algorithms. One of the main contributions to lattice reduction was the work of Lenstra, Lenstra, and Lovász in 1982 [LLL82], where they introduced the LLL algorithm, which was the first polynomial time algorithm to solve the approximate shortest vector problem in higher dimensions. Another famous contribution is the BKZ block algorithm of Schnorr and Euchner [SE91]. In practice, this is the algorithm that gives the best solution to lattice reduction so far. Their paper [SE91] also introduces the enumeration algorithm (ENUM), which practically is the fastest algorithm to solve the exact shortest vector problem using complete enumeration of all lattice vectors. It is used as a black box in the BKZ algorithm. The enumeration algorithm organizes linear combinations of the basis vectors in a search tree and performs a depth first search above the tree. In the same paper, Schnorr and Euchner explain the idea of deep inserting vectors into a basis during LLL reduction. This approach results in shorter vectors at the expense of the algorithms runtime.

Other promising algorithm variants were presented by Schnorr [Sch03], Nguyen and Stehlé [NS05], and Gama and Nguyen [GN08a]. The variant of [NS05] is implemented in the `fpLLL` library of [Ste], which is the fastest public implementation of ENUM algorithms. In [GN08b] Gama and Nguyen compare the NTL implementation [Sho] of floating point LLL, the deep insertion variant of LLL and the BKZ algorithm. It is the first comprehensive comparison of lattice basis reduction algorithms and helps understanding their practical behavior. Koy introduced the notion of a primal-dual reduction in [Koy04]. Schnorr [Sch03] and Ludwig [BL06] deal with random sampling reduction. Both are a bit different concepts of lattice reduction, where primal-dual reduction uses the dual of a lattice for reducing and random sampling combines LLL-like algorithms with an exhaustive point search in a set of lattice vectors that is likely to contain short vectors.

The parallelization of lattice reduction was considered in [Vil92,HT93,RV92]. These papers present parallel versions for n and n^2 processors, where n is the lattice dimension. In [Jou93] the parallel LLL of Villard [Vil92] is combined with the floating point ideas of [SE91]. In [Wet98] the authors present a blockwise

generalization of Villards algorithm. Backes and Wetzel worked out a parallel variant of the LLL algorithm for multi-core CPU architectures [BW09]. All previous parallel algorithms handle the LLL algorithm, but to our knowledge there exists no parallel version of the enumeration algorithm. Furthermore, for GPU parallelization there is no work done.

Applications. Lattice reduction has applications in cryptography as well as in cryptanalysis. The foundation of some promising cryptographic primitives is based on the hardness of lattice problems. Lattice reduction helps determining the practical hardness of those problems and is a basis for real world application of those hash functions, signatures, and encryption schemes. Well known examples are the SWIFFT hash functions of Lyubashevsky et al. [LMPR08], the signature scheme of Gentry, Peikert and Vaikuntanathan [GPV08], or the encryption schemes of [AD97,Pei09,SSTX09]. The NTRU [HPS98,otCC09] and GGH [GGH97] schemes do not provide a security proof, but the most promising attacks are also lattice based ones.

In cryptanalysis, there are further applications of lattice basis reduction. Not only lattice-based systems can be broken using this technique. There are attacks on RSA and similar systems, using lattice reduction to find roots of certain polynomials [CNS99, DN00]. Low density knapsack cryptosystems were successfully attacked with lattice reduction [LO85]. Other applications of lattice basis reduction are factoring numbers and computing discrete logarithms using diophantine approximations [Sch91]. In Operations Research, or generally speaking, discrete optimization, lattice reduction can be used to solve linear integer programs [Len83].

2.2 Programming Graphics Cards

Graphical Processing Units (GPUs) is hardware that is specifically designed to perform a massive number of specific graphical operations in parallel. The introduction of platforms like CUDA by NVidia [Nvi07a] or CTM by ATI [AMD06], that make it easier to run custom programs instead of limited graphical operations on a GPU, has been the major breakthrough for the GPU as a general computing platform. The introduction of integer and bit arithmetic also broadened the scope to cryptographic applications.

Applications. Many general mathematical packages are available for GPU, like the BLAS library [NVI07b] that supports basic linear algebra operations.

An obvious application in the area of cryptography is brute force searching using multiple parallel threads on the GPU. There are also implementations of AES [CIKL05] [Man07] [HW07] and RSA [MPS07] [SG08], [Fle07] available. These implementations can also be used (partially) in cryptanalysis. Integer factorization on elliptic curves has been implemented in [BCC⁺09]. However, to date, no applications based on lattices are available for GPU.

Programming Model. For the work in this paper the CUDA platform will be used. The GPUs from the Tesla range, which support CUDA, are composed of several multiprocessors, each containing a small number of scalar processors. For the programmer this underlying hardware model is hidden by the concept of SIMT-programming: Single Instruction, Multiple Thread. The basic idea is that the code for a single thread is written, which is then uploaded to the device and executed in parallel by multiple threads.

The threads are organized in multidimensional arrays, called blocks. All blocks are again put in a multidimensional array, called the *grid*. When executing a program (a grid), threads are scheduled in groups of 32 threads, called *warps*. Within a warp threads should not diverge, as otherwise the execution of the warp is serialized.

Memory Model. The Tesla GPUs provide multiple levels of memory: registers, shared memory, global memory, texture and constant memory. Registers and shared memory are on chip and close to the multiprocessor and can be accessed with low latency. The number of registers and shared memory is limited, since the number available for one multiprocessor must be shared among all threads in a single block.

Global memory is off-chip and is not cached. As such, access to global memory can slow down the computations drastically, so several strategies for speeding up memory access should be considered (besides the general strategy of avoiding global memory access). By coalescing memory access, e.g. loading the same memory address or a consecutive block of memory from multiple threads, the delay is reduced, since a coalesced memory access has the same cost as a single random memory access. By launching a large number of blocks the latency introduced by memory loading can also be hidden, since other blocks can be scheduled in the meantime.

The constant and texture memory are cached and can be used for specific types of data or special access patterns.

Instruction Set. Modern GPUs provide the full range of (32 and) 64 bit floating point, integer and bit operations. Addition and multiplication are fast, other operations can, depending on the type, be much slower. There is no point in using other than 32 or 64 bit numbers, since smaller types are always cast to larger types. Most GPUs have a specialized FMAD instruction, which performs a floating point multiplication followed by an addition at the cost of only a single operation. This instruction can be used during the BKZ enumeration.

One problem that occurs on GPU's is the fact that today GPU's are not able to deal with higher precision than 64 bit floating point numbers. For lattice reduction, sometimes higher bit sizes are required to guarantee the correct termination of the algorithms. For an n -dimensional lattice, using the floating point LLL algorithm of [LLL82], one requires a precision of $\mathcal{O}(n \log B)$ bits, where B is an upper bound for the d -dimensional vectors [NS05]. For the L^2 algorithm of [NS05], the required bit size is $\mathcal{O}(n \log_2 3)$, which is independent of the entry size. For more details on the floating point LLL analysis see [NS05] and [NS06].

In [PS08] the authors state that for enumeration algorithms double precision is suitable up to dimension 90, which is beyond the dimensions that are practical today. Therefore enumeration should be possible on actual graphics cards, whereas LLL-like algorithms will gain some problems and require some multi-precision framework.

3 Parallel Enumeration on GPU

In this section we present our algorithm for shortest vector enumeration in lattices. Firstly we briefly explain the ENUM algorithm of Schnorr and Euchner [SE91]. Secondly we explain the ideas of GPU parallelization of the algorithm before presenting our new algorithm.

The enumeration algorithms used are variants of those in [Kan83] and [FP83]. Schnorr and Euchner [SE91] improve the enumeration technique. Their algorithm is the fastest one today and also the one used in the NTL [Sho] and `fpLLL` [Ste] library. Therefore we have chosen this algorithm as basis for our parallel algorithm.

The ENUM algorithm enumerates over all linear combinations $[x_1, \dots, x_n] \in \mathbb{Z}^n$ which generate a vector $\mathbf{v} = \sum_{i=1}^n x_i \mathbf{b}_i$. Those linear combinations are organized in a tree structure. Leafs of the tree contain full linear combinations, whereas inner nodes contain partly filled vectors. The search for the tree leaf that determines the shortest lattice vector is performed in a depth first search order. The most important part of the enumeration is cutting off parts of the tree, i.e. the strategy which subtrees are explored and which ones cannot lead to a shorter vector. Usually, as initial bound for the length of the shortest vector, one uses the norm of the first basis vector. For a more detailed description we refer to [PS08].

3.1 Multi-Thread Enumeration

Roughly speaking, the parallel enumeration works as follows. The search tree of combinations that is explored in the enumeration algorithm can be split at a high level, distributing subtrees among several threads. Each thread then runs an enumeration algorithm, keeping the first coefficients fixed. These threads can run independently of the others, which limits communication needed between threads. The top level enumeration is performed on CPU and outputs start vectors for the single GPU threads.

When the number of postponed subtrees is higher than the number of threads that we can start in parallel, then we copy the start vectors to the GPU and let it enumerate the subtrees. After all threads have finished enumerating their subtrees we proceed in the same manner: caching start vectors on CPU and start enumeration on GPU. Figure 1 illustrates this approach. The variable α defines the region where the initial enumeration is performed. The subtrees where GPU threads work are also depicted in Figure 1.

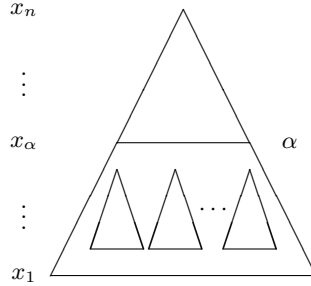


Fig. 1. Illustration of the algorithm flow. The top part is enumerated on CPU, the lower subtrees are explored in parallel on GPU.

If a GPU subtree enumeration finds a new optimal vector, it writes back the coordinates and norm of this vector to the main memory. The other GPU threads will directly receive the new norm, which will allow them to cut away more parts of the subtree.

Early Termination. The computation power of the GPU is used best when as many threads as possible are working at the same time. Therefore the number of enumeration steps that can be performed on GPU is bounded by a value \mathbf{S} (for each subtree). When \mathbf{S} is exceeded by one of the GPU subtree enumerations, this subtree enumeration stops computing and writes back its current state to the main memory. This state can be used later on, to restart the enumeration at exactly the same position it stopped. The early termination after \mathbf{S} enumeration steps is needed because the subtree enumerations have different lengths. The early termination ensures that the GPU is always running a high number of enumerations in parallel and isn't stalled by a small number of long-running enumerations. This small number of long-running threads are the cause of thread divergence, which causes performance degradation.

Because of early termination some of the subtree enumerations are not finished after a single launch of the GPU enumeration. This is the main reason why the entire algorithm is iterated several times. At each iteration the GPU launches a mix of enumerations: new subtrees from the top enumeration and subtrees that were not finished in one of the previous GPU launches (because \mathbf{S} was exceeded).

3.2 The Iterated Parallel ENUM Algorithm

Algorithm 1 shows the high-level layout of the GPU enumeration algorithm. Details concerning the updating of the bound A , as well as the write-back of newly discovered optimal vectors have been omitted. The actual enumeration is also not shown: it is part of several subroutines which are called from the main algorithm.

Algorithm 1: High-level GPU ENUM Algorithm

Input: $\mathbf{b}_i, A, \alpha, n$

```

1 Compute the Gram-Schmidt decomposition of  $\mathbf{b}_i$ 
2 while true do
3    $S = \{(\mathbf{x}_i, \Delta\mathbf{x}_i, \Delta^2\mathbf{x}_i, l_i = \alpha, s_i = 0)\}_i \leftarrow$  Top enum: generate at most
   NUMSTARTPOINTS  $- \#T$  vectors
4    $R = \{(\bar{\mathbf{x}}_i, \Delta\mathbf{x}_i, \Delta^2\mathbf{x}_i, l_i, s_i)\}_i \leftarrow$  GPU enumeration, starting from  $S \cup T$ 
5    $T \leftarrow \{R_i : s_i \geq \mathbf{S}\}$ 
6   if  $\#T < \text{CPUTHRESHOLD}$  then
7     Enumerate the starting points in  $T$  on the CPU.
8     Stop
9   end
10 end

```

Output: (x_1, \dots, x_n) with $\|\sum_{i=1}^n x_i \mathbf{b}_i\| = \lambda_1(L)$

The whole process of launching a grid of GPU threads is iterated several times (line 2), until the whole search tree has been enumerated either on GPU or CPU.

In line 3, the top of the search tree is enumerated, to generate a set S of starting vectors \mathbf{x}_i for which enumeration should be started at level α . More detailed, the top enumeration in the region between α and n outputs distinct vectors

$$\mathbf{x}_i = [0, \dots, 0, x_\alpha, \dots, x_n] \quad \text{for } i = 1 \dots \text{NUMSTARTPOINTS} - \#T.$$

The top enumeration will stop automatically if a sufficient amount of vectors from the top of the tree have been enumerated. The rest of the top of the tree is enumerated in the following iterations of the algorithm.

Line 4 performs the actual GPU enumeration. In each iteration, a set of starting vectors and starting levels $\{\mathbf{x}_i, l_i\}$ is uploaded to the GPU. These starting vectors can be either vectors generated by the top enumeration in the region between α and n (in which case $l_i = \alpha$) or the vectors (and levels) written back by the GPU when exceeding \mathbf{S} , so that the enumeration will continue. In total NUMSTARTPOINTS vectors (a mix of new and old vectors) are uploaded at each iteration. For each starting vector \mathbf{x}_i (with associated starting level l_i) the GPU outputs a vector

$$\bar{\mathbf{x}}_i = [\bar{x}_1, \dots, \bar{x}_{\alpha-1}, x_\alpha, \dots, x_n] \quad \text{for } i = 1 \dots \text{NUMSTARTPOINTS},$$

(which describes the current position in the search tree), the current level l_i , the number of enumeration steps s_i performed and also part of the internal state of the enumeration. This state $\{\bar{\mathbf{x}}_i, \Delta\mathbf{x}_i, \Delta^2\mathbf{x}_i, l_i\}$ can be used to continue the enumeration later on. The vectors $\Delta\mathbf{x}_i$ and $\Delta^2\mathbf{x}_i$ are used in the enumeration to generate the zig-zag pattern and are part of the internal state of the enumeration [SE91]. This state is added to the output to be able to efficiently restart the enumeration at the point it was terminated.

Line 5 will select the resulting vectors from the GPU enumeration that were terminated because of reaching the bound \mathbf{S} . These will be added to the set T of *leftover* vectors, which will be relaunched in the next iteration of the algorithm. If the set of leftover vectors is too small to get an efficient GPU enumeration, the CPU takes over and finishes of the last part of the enumeration. This final part only takes limited time.

GPU Threads. The number of starting points `NUMSTARTPOINTS` is higher than the number of threads (`NUMTHREADS`) that are launched on the GPU. Each thread will pick a new starting vector from the stack, after it finished enumerating his starting point. This is done until all `NUMSTARTPOINTS` starting vectors are enumerated or stopped after reaching \mathbf{S} enum steps. The reason for this is closely connected to the early termination feature discussed before. Since the subtree enumerations have different lengths, a thread should be able to continue working even if the subtree enumeration was small. In our experiments, `NUMSTARTPOINTS` was around 20-30 times higher than `NUMTHREADS`, which means that on average every GPU thread enumerated 20-30 subtrees in each iteration.

4 Experimental Results

In this section we present some preliminary results of our CUDA implementation of our algorithm. For comparison we used the highly optimized ENUM algorithm of the `fpLLL` library in version 3.0.11 of Stehlé and Pujol from [Ste]³. The CUDA program was compiled using `nvcc`, for the CPU programs we used `g++` with compiler flag `-O2`. The tests were run on a Intel Core2 Extreme CPU X9650 (using one single core) running at 3 GHz, and a NVIDIA GTX 280 graphics card. We run up to 100000 threads in parallel on the GPU.

Our input lattices are LLL reduced with $\delta = 0.99$. We chose random lattices following the construction principle of [GM03] with bit size of the entries of $10 \cdot n$. Both algorithms output the same coefficient vectors and therefore a lattice vector with shortest possible length. Table 1 and Figure 2 illustrate the experimental

n	45	48	50	52	54
fpLLL	18.3s	139s	277s	2483s	6960s
CUDA	20.2s	92s	133s	959s	2599s
	110%	66%	48%	39%	37%

Table 1. Average time needed for enumeration of lattices in each dimension n .

results. They compare our CUDA implementation with the ENUM of `fpLLL` (run `fpLLL` with parameter `-a svp`). The figure shows the runtimes of both algorithms when applied to five different lattices of each dimension. One can notice that in

³ Timings of the NTL version of ENUM can be found in [GN08b].

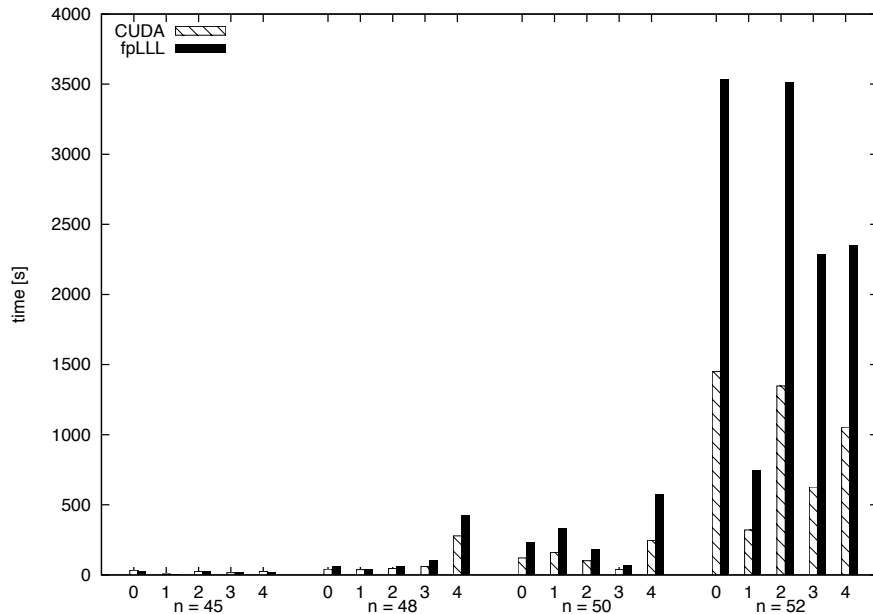


Fig. 2. Timings for enumeration. The graph shows the time needed for enumerating five different random lattices in each dimension n .

dimension above 48, our CUDA implementation always outperforms the `fpLLL` implementation. In lower dimensions, the initialization of the GPU requires more time than the enumeration itself, therefore the complete algorithm lasts longer than on CPU. Table 1 shows the average value over all five lattices in each dimension. Again one notices that in dimension 45, the GPU algorithm is a bit slower. It demonstrates its strength in dimensions above 48, where the time goes down to 48% in dimension 50 and down to 37% in dimension 54. Therefore we can state that the GPU algorithm gains big speedups in dimensions bigger than 50, which are the interesting ones in practice.

On the GPU a throughput up to 100 million enumeration steps per second is achieved, while similar experiments on CPU only yielded 25 million steps per second.

Further Work. Further improvements are possible using multiple CPU cores. Our implementation only uses one CPU core for the top enumeration, whereas additional cores are kept aside for the computations. When GPU starts enumeration it would be possible to start threads on different CPU cores as well. We expect a speedup of two compared to our actual implementation using this idea.

A second opportunity for further speedups is the tweaking of the several parameters (like \mathbf{S} , α , `NUMTHREADS`, `NUMSTARTPOINTS`). Exhaustive testing will improve the sets of parameters for GPU enumeration for each lattice dimension.

It is possible to start enumeration using a shorter starting value than the first basis vectors norm. The Gaussian heuristic can be used to predict the norm of the shortest basis vector λ_1 . This can lead to enormous speed ups in the algorithm. We did not include this improvement into our algorithm so far to get comparable results to `fpLLL`.

Acknowledgments

We thank the anonymous referees for their valuable comments. We thank Özgür Dagdelen for creating some of the initial ideas of parallelizing lattice enumeration and Benjamin Milde for the nice discussions and help with the implementation.

Finally we would like to thank EcryptII⁴ and CASED⁵ for providing the funding for the visits during which this work was prepared.

References

- [AD97] Miklós Ajtai and Cynthia Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *Proceedings of the Annual Symposium on the Theory of Computing (STOC) 1997*, pages 284–293, 1997.
- [AMD06] Advanced Micro Devices. ATI CTM Guide. Technical report, 2006.
- [BCC⁺09] Daniel J. Bernstein, Tien-Ren Chen, Chen-Mou Cheng, Tanja Lange, and Bo-Yin Yang. ECM on graphics cards. In *Advances in Cryptology — Eurocrypt 2009*, Lecture Notes in Computer Science, pages 483–501, 2009.
- [BL06] Johannes Buchmann and Christoph Ludwig. Practical lattice basis sampling reduction. In F. Hess, S. Pauli, and M. Pohst, editors, *Proceedings of ANTS VII*, volume 4076 of *Lecture Notes in Computer Science*, pages 222–237. Springer-Verlag, 2006.
- [BW09] Werner Backes and Susanne Wetzels. Parallel lattice basis reduction using a multi-threaded Schnorr-Euchner LLL algorithm. In *15th International European Conference on Parallel and Distributed Computing (Euro-Par)*, 2009.
- [CIKL05] Debra L. Cook, John Ioannidis, Angelos D. Keromytis, and Jake Luck. Cryptographics: Secret key cryptography using graphics cards. In *CT-RSA*, pages 334–350, 2005.
- [CNS99] Christophe Coupé, Phong Q. Nguyen, and Jacques Stern. The effectiveness of lattice attacks against low exponent RSA. In *Public-Key Cryptography (PKC)*, volume 1560 of *Lecture Notes in Computer Science*, pages 204–218. Springer-Verlag, 1999.
- [DN00] Glenn Durfee and Phong Q. Nguyen. Cryptanalysis of the RSA schemes with short secret exponent from Asiacrypt '99. In *Advances in Cryptology — Asiacrypt 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 14–29. Springer-Verlag, 2000.
- [ECR] ECRYPT II. Wiki - Hard problems in cryptography - lattices. <http://www.ecrypt.eu.org/wiki/index.php/Lattices>.

⁴ <http://www.ecrypt.eu.org/>

⁵ <http://www.cased.de>

- [Fle07] S. Fleissner. GPU-Accelerated Montgomery Exponentiation. *Lecture Notes in Computer Science*, 4487:213, 2007.
- [FP83] U. Fincke and Michael Pohst. A procedure for determining algebraic integers of given norm. In *European Computer Algebra Conference*, volume 162 of *Lecture Notes in Computer Science*, pages 194–202. Springer-Verlag, 1983.
- [GGH97] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Public-key cryptosystems from lattice reduction problems. In *Advances in Cryptology — Crypto 1997*, Lecture Notes in Computer Science, pages 112–131. Springer-Verlag, 1997.
- [GM03] Daniel Goldstein and Andrew Mayer. On the equidistribution of hecke points. *Forum Mathematicum 2003*, 15:2, pages 165–189, 2003.
- [GN08a] Nicolas Gama and Phong Q. Nguyen. Finding short lattice vectors within mordell’s inequality. In *Proceedings of the Annual Symposium on the Theory of Computing (STOC) 2008*, pages 207–216. ACM Press, 2008.
- [GN08b] Nicolas Gama and Phong Q. Nguyen. Predicting lattice reduction. In *Advances in Cryptology — Eurocrypt 2008*, pages 31–51, 2008.
- [GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the Annual Symposium on the Theory of Computing (STOC) 2008*, pages 197–206. ACM Press, 2008.
- [HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In *Algorithmic Number Theory Symposium — ANTS 1998*, pages 267–288, 1998.
- [HT93] Christian Heckler and Lothar Thiele. A parallel lattice basis reduction for mesh-connected processor arrays and parallel complexity. In *SPDP*, pages 400–407, 1993.
- [HW07] O. Harrison and J. Waldron. AES Encryption Implementation and Analysis on Commodity Graphics Processing Units. *Lecture Notes in Computer Science*, 4727:209, 2007.
- [Jou93] Antoine Joux. A fast parallel lattice reduction algorithm. In *Proceedings of the Second Gauss Symposium*, pages 1–15, 1993.
- [Kan83] Ravi Kannan. Improved algorithms for integer programming and related lattice problems. In *Proceedings of the Annual Symposium on the Theory of Computing (STOC)1983*, pages 193–206. ACM Press, 1983.
- [Koy04] Henrik Koy. Primale-duale Segment-Reduktion. <http://www.mi.informatik.uni-frankfurt.de/research/papers.html>, 2004.
- [Len83] H.W.jun. Lenstra. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8:538–548, 1983.
- [LLL82] Arjen Lenstra, Hendrik Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.
- [LMPR08] Vadim Lyubashevsky, Daniele Micciancio, Chris Peikert, and Alon Rosen. Swift: A modest proposal for fft hashing. In *Fast Software Encryption (FSE) 2008*, Lecture Notes in Computer Science, pages 54–72. Springer-Verlag, 2008.
- [LO85] J. C. Lagarias and Andrew M. Odlyzko. Solving low-density subset sum problems. *J. ACM*, 32(1):229–246, 1985.
- [Man07] S.A. Manavski. Cuda Compatible GPU as an Efficient Hardware Accelerator for AES Cryptography. 2007.
- [MPS07] A. Moss, D. Page, and N.P. Smart. Toward Acceleration of RSA Using 3D Graphics Hardware. *Lecture Notes in Computer Science*, 4887:364, 2007.

- [NS05] Phong Q. Nguyen and Damien Stehlé. Floating-point LLL revisited. In *Advances in Cryptology — Eurocrypt 2005*, pages 215–233, 2005.
- [NS06] Phong Q. Nguyen and Damien Stehlé. LLL on the average. In *Algorithmic Number Theory Symposium — ANTS 2006*, pages 238–256, 2006.
- [Nvi07a] Nvidia. Compute Unified Device Architecture Programming Guide. Technical report, 2007.
- [NVI07b] NVIDIA. CUBLAS Library, 2007.
- [otCC09] 1363 Working Group of the C/MM Committee. IEEE P1363.1 Standard Specification for Public-Key Cryptographic Techniques Based on Hard Problems over Lattices, 2009. Available at <http://grouper.ieee.org/groups/1363/>.
- [Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In *Proceedings of the Annual Symposium on the Theory of Computing (STOC) 2009*, pages 333–342, 2009.
- [PS08] Xavier Pujol and Damien Stehlé. Rigorous and efficient short lattice vectors enumeration. In *Advances in Cryptology — Asiacrypt 2008*, pages 390–405, 2008.
- [RV92] J. L. Roch and G. Villard. Parallel gcd and lattice basis reduction. In L. Bougé, M. Cosnard, Y. Robert, and D. Trystram, editors, *Proceedings of the Second Joint International Conference on Vector and Parallel Processing. Parallel Processing: VAPP V, CONPAR'92 (Lyon, France, September 1992)*, volume 634 of *Lecture Notes in Computer Science*, pages 557–564. Springer-Verlag, 1992.
- [Sch91] Claus-Peter Schnorr. Factoring integers and computing discrete logarithms via diophantine approximations. In *Advances in Cryptology — Eurocrypt 1991*, pages 281–293, 1991.
- [Sch03] Claus-Peter Schnorr. Lattice reduction by random sampling and birthday methods. In *STACS 2003: 20th Annual Symposium on Theoretical Aspects of Computer Science*, volume 2607 of *Lecture Notes in Computer Science*, pages 146–156. Springer-Verlag, 2003.
- [SE91] Claus-Peter Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. In *FCT '91: Proceedings of the 8th International Symposium on Fundamentals of Computation Theory*, pages 68–85. Springer-Verlag, 1991.
- [SG08] R. Szerwinski and T. Guneyusu. Exploiting the Power of GPUs for Asymmetric Cryptography. *Lecture Notes in Computer Science*, 5154:79–99, 2008.
- [Sho] Victor Shoup. Number theory library (NTL) for C++. <http://www.shoup.net/ntl/>.
- [SSTX09] Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. Efficient public key encryption based on ideal lattices. Cryptology ePrint Archive, Report 2009/285, 2009. <http://eprint.iacr.org/>.
- [Ste] Damien Stehlé. Damien Stehlé's homepage at école normale supérieure de Lyon. <http://perso.ens-lyon.fr/damien.stehle/english.html>.
- [Vil92] Gilles Villard. Parallel lattice basis reduction. In *ISSAC*, pages 269–277, 1992.
- [Wet98] Susanne Wetzel. An efficient parallel block-reduction algorithm. In J. P. Buhler, editor, *Proceedings of the 3rd International Symposium on Algorithmic Number Theory, ANTS'98 (Portland, Oregon, June 21-25, 1998)*, volume 1423 of *Lecture Notes in Computer Science*, pages 323–337. Springer-Verlag, 1998.