IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 10, 2019, accepted August 22, 2019, date of publication September 3, 2019, date of current version September 18, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2939142

Heterogeneous Implementation of a Voronoi Cell-Based SVP Solver

GABRIEL FALCAO[1], (Senior Member, IEEE), FILIPE CABELEIRA[1], ARTUR MARIANO[2], AND LUIS PAULO SANTOS[2,3]

[1]Department of Electrical and Computer Engineering, Instituto de Telecomunicações, University of Coimbra, 3030-290 Coimbra, Portugal
[2]INESC TEC, 4200-465 Porto, Portugal
[3]Departamento de Informática, Universidade do Minho, 4710-057 Braga, Portugal

Corresponding author: Gabriel Falcao (gff@co.it.pt)

This work was supported in part by the Instituto de Telecomunicações, in part by the Fundação para a Ciência e a Tecnologia (FCT) under Grant UID/EEA/50008/2019 and Grant PTDC/EEI-HAC/30485/2017, and in part by the National Funds through the Portuguese Funding Agency, FCT—Fundação para a Ciência e a Tecnologia, under Grant UID/EEA/50014/2019. The work of A. Mariano was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Grant 382285730.

ABSTRACT This paper presents a new, heterogeneous CPU+GPU attacks against lattice-based (post-quantum) cryptosystems based on the Shortest Vector Problem (SVP), a central problem in lattice-based cryptanalysis. To the best of our knowledge, this is the first SVP-attack against lattice-based cryptosystems using CPUs and GPUs simultaneously. We show that Voronoi-cell based CPU+GPU attacks, algorithmically improved in previous work, are suitable for the proposed massively parallel platforms. Results show that 1) heterogeneous platforms are useful in this scenario, as they increment the overall memory available in the system (as GPU's memory can be used effectively), a typical bottleneck for Voronoi-cell algorithms, and we have also been able to increase the performance of the algorithm on such a platform, by successfully using the GPU as a co-processor, 2) this attack can be successfully accelerated using conventional GPUs and 3) we can take advantage of multiple GPUs to attack lattice-based cryptosystems. Experimental results show a speedup up to 7.6× for 2 GPUs hosted by an Intel Xeon E5-2695 v2 CPU (12 cores ×2 sockets) using only 1 core and gains in the order of 20% for 2 GPUs hosted by the same machine using all 22 CPU threads (2 are reserved for orchestrating the GPUs), compared to single-CPU execution using the entire 24 threads available.

INDEX TERMS Lattices, lattice-based cryptanalysis, Voronoi-cell, algorithms, high performance computing, parallelism, multi-threading, multicores, graphics processing units, multi-GPU, parallel computing, CUDA, OpenMP, StarPU.

I. INTRODUCTION

Two decades ago, it was shown that quantum computers will easily break current cryptosystems. With the discovery of polynomial time algorithms, solving the underlying mathematical problems, such as factorization of large numbers and the computation of discrete logarithms becomes far simpler [8], [42], [43]. Since then, finding efficient alternatives to classical cryptosystems, such as RSA and ElGamal, has become a central goal for the cryptography scientific community.

The associate editor coordinating the review of this article and approving it for publication was Gang Mei.

A. PREPARING FOR THE POST-QUANTUM ERA

As time went by, several cryptosytems have been proposed [4], [10], [15], [19], [20], in order to anticipate the rise of this so-called post-quantum era. In fact, this has been a race against the clock, as it urges to find efficient and safe alternatives before quantum computers are practical. Therefore, all the proposed cryptosytems are based on the premise that adversaries cannot brake them, even if they have access to large-scale quantum computers. This is typically impossible to prove and empirical attacks have to be conducted, so that security parameters are defined based on the empirical performance of the best attacks, implemented on the best computing platforms. Until quantum computers are available, we are forced to draw conclusions based on the

best computing platforms available today, such as massively parallel platforms. Up until now, there is substantial work on computing platforms with tens of CPU cores e.g. [9], [13], [30], or, alternatively, solely on GPUs, but there is no work on CPU+GPU platforms, except for building blocks for attacks rather than attacks themselves [23] or attacks that can be broken down into many instances of the same attack [25], which is typically not the case with most attacks.

B. LATTICE-BASED CRYPTOSYSTEMS

Currently, lattice-based cryptosystems constitute the most promising type of post-quantum cryptosystems; first, they support Fully Homomorphic Encryption [18], which allows any operation on encrypted data without decrypting it, second, they are relatively efficient in practice [4], [33], and third, they are easy to implement [22], [31], [32]. Needless to say, they are also believed to be safe against quantum adversaries [8], [33]. The key concept of most modern and ancient cryptosystems is a mathematical problem that is easily solved by the parties using the cryptosystem but very hard to solve by eavesdroppers. For lattice-based cryptosystems, those problems are the Shortest Vector Problem (SVP), the Closest Vector Problem (CVP) and derivatives of these. The reason why lattice-based cryptosystems are believed to be secure for the post-quantum era is that these problems cannot be solved (exponentially) faster with quantum computers, when compared to conventional computers. In this paper, we refer to the algorithms that solve these problems as *attacks*.

C. LATTICES

As lattices are the backbone of lattice-based cryptosystems, it is important to explain this concept.

Lattices are discrete subgroups of the n-dimensional Euclidean space $\mathbb{R}^n$, with a strong periodicity property. We refer the reader to the papers [35], [39] in order to learn more about lattices, especially in the context of lattice-based cryptography.

A lattice $\mathcal{L}$ generated by a basis $\mathbf{B}$, a set of linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_m$ in $\mathbb{R}^n$, is denoted by:

$$\mathcal{L}(\mathcal{B}) = \left\{ x \in \mathbb{R}^n : x = \sum_{i=1}^{m} u_i b_i,\ u \in \mathbb{Z}^m \right\}, \qquad (1)$$

where $m \leq n$ is the *rank* of the lattice. When $m = n$, the lattice is said to be of *full rank*. When n is at least 2, each lattice has infinitely many different bases.

Although there are non-integer lattices, lattice-based cryptography commonly uses integer lattices in practice: solving lattice problems on integer lattices is still hard, and integer lattices are easier to handle computationally (e.g. there are no precision/numerical problems). As an example, Figure 1 shows a lattice in $\mathbb{R}^2$, where the basis is $B = \{b_1, b_2\}$. The vector b_3 shown in the picture is a linear combination of the basis vectors. This linear combination also shows that b_1 can be made shorter (in terms of Euclidean norm, which is the default meaning of length in the context of this paper) at the cost of b_2, given that b_3 is smaller than b_1. This process, of making lattice vectors (bases) shorter by adding/subtracting other lattice vectors, is often referred to as vector (basis) reduction, and is widely used in various lattice algorithms.

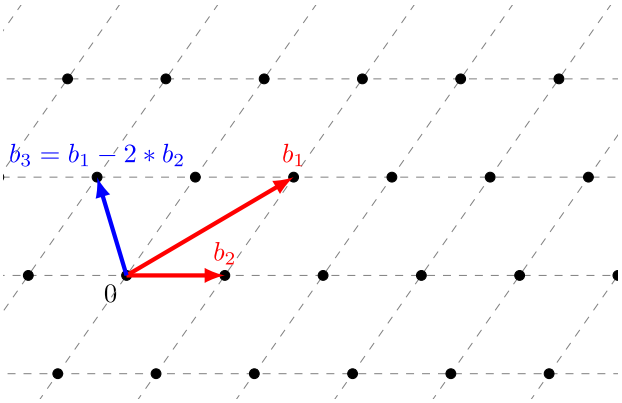

FIGURE 1. Example of a lattice in $\mathbf{R^2}$ and its basis ($\mathbf{b_1}$,$\mathbf{b_2}$) in red.

D. LATTICE PROBLEMS FOR LATTICE-BASED CRYPTOGRAPHY

The security of lattice-based cryptosystems is based on problems like the SVP, CVP and approximated versions of these. These problems have been widely studied over the last decades, especially from a theoretical standpoint e.g. [2], [3], [17], [26], [34]. The work around implementations of these problems has also been getting traction over the last decade. In particular, many parallel, efficient versions of algorithms that solve these problems have been proposed e.g. [9], [13], [21], [30]. As mentioned, this is key to assess the security of the cryptosystems, by assessing the real hardness of SVP and other problems that underpin the security of the cryptosystems. The key idea is that cryptosystems have the so called "security parameters" (e.g. the key size), which cannot be predicted upfront in order to make the system secure. Instead, we rely on the performance and potential of the best attacks, in order to extrapolate such parameters. Intuitively, we would simply set these parameters very high, but the problem is that this leads to inefficient (i.e. slow) cryptosystems (and setting them too low results in insecure systems). Therefore, only practical testing with attacks enables us to choose realist, grounded parameters that are both efficient and secure. To this end, attacks must leverage all the computing power available on the system, which often goes well beyond the CPU, including co-processors initially dedicated to specific tasks, such as the GPUs. As the implementation of algorithms on heterogenous CPU+GPU architectures poses many challenges, including e.g. data management and workload distribution, this is usually done with frameworks that assist programmers in this task. In this paper, we used StarPU [6], a task-based framework for CPU+GPU architectures, which we briefly present later on. As we show in this paper (Section V explains this in detail), we have identified a gap in the available work: there is very little exploration of heterogeneous platforms in

the context of lattice-based cryptosystems and none pertaining to hard attacks, such as those based on the SVP, which we solve in this paper.

E. OUR CONTRIBUTION

In this paper, we select a type of attack – an SVP-solver based on the Voronoi cell of a lattice – that has been often mentioned in the literature [2], [12], [34], but rarely studied or published about, as this algorithm is commonly accepted as impractical. In this work, we show that the algorithm can be made practical while taking advantage of massively parallel architectures, in particular those with CPUs and GPUs. This work is also innovative as we present the first ever heterogeneous implementation in the context of lattice-based cryptanalysis and the SVP in particular. There are two main motivations for this work: 1) although we are aware of this algorithm practical limitations, due to increasing memory requirements with the number of dimensions, the actual performance of Voronoi cell was never studied in-depth, something we address in this paper and 2) this is the first paper on a single instance of a SVP attack implemented on a CPU+GPU platform, providing novel insight on how to make use of these platforms in the context of lattice-based cryptanalysis. Implementing this algorithm on heterogeneous environments makes sense at several levels, one of which being the possibility to harness the entire memory space in the hardware, a remarkable problem in Voronoi cell-based algorithms. Our optimizations on this algorithm, together with the heterogeneous implementation that we devised, show that the algorithm is far more practical than previously thought. We hope that this paper opens up a new realm of research around Voronoi cell-based algorithms for lattice-based cryptanalysis.

F. ROADMAP

Section II shows the specifications of the hardware we used to conduct the experiments reported on this paper. Section III presents the algorithms, including the original Voronoi algorithm [2] and its upgraded version, Voronoi 2.0 Section IV briefly presents a CUDA implementation of Voronoi 2.0, which kickstarted our work. Section V lays out the reasons for using CPU+GPU platforms for this problem. Section VI goes over our heterogeneous implementation, briefly presenting StarPU, which underpins our implementation, and the several aspects of the heterogeneous implementation, including workload distribution, scheduling and data management. It also shows the performance of our heterogeneous implementation in practice, whose results we comment on. Section VII shows the practical impact of our contributions and Section VIII concludes the paper.

II. HARDWARE SPECIFICATIONS

To carry out the tests in this paper, we picked the machines specified in Table 1.

The clock frequency in parenthesis is the maximum frequency of the CPU, when Turbo Boost is turned on. L1 cache values are split between instruction cache (i) and data cache (d). Machine A runs Ubuntu 16.04.1 x86_64 with kernel version 4.13 and Machine B runs CentOS x86_64 with kernel version 2.6.32. All programs were compiled with the `-march=native -O3` optimization flags.

TABLE 1. Specifications of our computer systems. SMT stands for simultaneous multi-threading and HT stands for hyper threading.

Machine	A	B
Sockets	1	2
CPU	Intel Core i3 6100	Intel Xeon E5-2695 v2
Clock frequency	3.70 GHz	2.40 GHz (3.20 GHz)
Cores per socket	2	12
SMT	Yes (w/HT, 4 threads)	Yes (w/HT, 24 threads)
L1 Cache	32 kB i + 32 kB d	32 kB i + 32 kB d
L2 Cache	256 kB	256 kB
L3 Cache	3 MB	30 MB
RAM	8 GB	64 GB
Number of GPUs	1	2
GPU	NVIDIA GeForce 1060 GTX	NVIDIA Tesla K20m
GPU Clock rate	1759 MHz	706 MHz
GPU RAM	6 GB	5 GB
CPU/OpenMP Compiler	GCC (g++) 5.4.0	GCC (g++) 4.4.6
CUDA Compiler	CUDA Toolkit 9.1 Compiler (nvcc)	CUDA Toolkit 7.0.28 Compiler (nvcc)

The lattices used in this paper were obtained using the SVP-Challenge (https://www.latticechallenge.org/svp-challenge/) lattice basis generator.

III. THE ALGORITHM(S)

A. VORONOI CELL BY AGRELL ET AL.

The algorithm we implemented on our CPU+GPU computing platform, called Relevant vectors was presented by Agrell et al. in [2]. In essence, this algorithm is an enumeration-based CVP solver that can be adapted to solve the SVP, compute the Voronoi cell of a lattice, and other lattice problems. Formally, the Voronoi cell $\mathcal{V}$ of a lattice $\mathcal{L}$ – the set of all points closest to zero than any other lattice point – is given by Equation 2,

$$\mathcal{V}(\mathcal{L}) = \{x \in \mathbb{R}^n : ||x|| \leq ||x - v|| \quad \forall\, v \in \mathcal{L}\}, \qquad (2)$$

where n is the dimension of the lattice.

The idea behind enumeration algorithms is to examine all possible lattice points inside a certain radius – either from an hypersphere or a parallelepiped (the Voronoi-cell algorithm is based on the former).

The *Relevant Vectors* algorithm uses this enumeration-based CVP-solver to compute the Voronoi cell of a lattice. We can break down this algorithm into 4 steps:

- first, the algorithm generates the target vectors, which will later feed the enumeration-based CVP-solver;
- second, the lattice basis is converted to a lower triangular form (that exists for all lattice bases), required by the CVP-solver, so that lattices of any type are supported; this is equivalent to a change in coordinate system. The target vectors are also transformed into the new coordinate system.

- third, the CVP-solver (please see Algorithm Decode in paper [2]) is executed over all target vectors; this is referred to as the decode step;
- fourth, the output of the decode function is converted back to the original coordinate system and processed. If the result is valid, i.e. the decoded vector is indeed a relevant vector, it is added to the list of relevant vectors.

The solution to the SVP is the shortest vector of the Voronoi relevant vectors (set N in Algorithm 1). The pseudo-code of the implementation is shown in Algorithm 1.

Algorithm 1 Relevant Vectors

Input: Basis matrix $\mathbf{B}$
Output: Relevant Vectors $\mathcal{N}$

1 $\mathbf{M} = \text{Reduce}(\mathbf{B})$; `/* for example, using the LLL algorithm */`
2 $[\mathbf{Q}, \mathbf{R}] =$ QR decomposition of $\mathbf{M}$;
3 $\mathbf{G} = \mathbf{R}^T$;
4 $\mathbf{H} = \mathbf{G}^{-1}$; `/* lattice basis on modified coordinate system */`
5 $\mathcal{N} = \varnothing$; `/* list of relevant vectors */`
6 $\mathcal{TV} = \text{Compute}(\mathcal{TV})$; `/* see (3) */`
7 **forall the** *vectors* $\mathbf{s} \in \mathcal{TV}$ **do**
8 $\quad \mathcal{X} = \text{AllClosestPoints}(\mathbf{M}, H, Q, \mathbf{s})$;
9 $\quad$ **if** $|\mathcal{X}| = 2$ **then**
10 $\quad\quad \mathcal{N} = \mathcal{N} \bigcup \{2\mathbf{x} - 2\mathbf{s} : \mathbf{x} \in \mathcal{X}\}$;
11 **return** $\mathcal{N}$

Function AllClosestPoints

Input: Matrix $\mathbf{M}$, matrix H, matrix Q, vector $\mathbf{s}$
Output: List of vectors $\mathcal{X}$

1 Compute $\mathbf{x} = \mathbf{s}\mathbf{Q}^T$; `/* conversion of the target vector to the modified coordinate system */`
2 $\mathcal{U} = \text{Decode}(\mathbf{H}, \mathbf{x})$;
3 Compute γ as the lowest value $||\mathbf{uM} - \mathbf{s}||$ for all $\mathbf{u} \in \mathcal{U}$;
4 Compute $\mathcal{X}$ as all $\{\mathbf{uM} : \mathbf{u} \in \mathcal{U}, ||\mathbf{uM} - \mathbf{s}|| = \gamma\}$
5 **return** $\mathcal{X}$

As shown in Line 1 of Algorithm 1, it is desirable to use a lattice basis reduction algorithm on the input basis $\mathbf{B}$, resulting in the reduced basis $\mathbf{M}$ (this could either be done with LLL [27] or BKZ [11], [41]), as SVP-solvers find shortest vectors faster on reduced bases. This increases both performance and numerical stability, given that lattice basis reduction algorithms shorten the basis vectors.

After reducing the input basis, the algorithm performs the coordinate system transformation. Given this is constant for a given lattice, it is only performed once. This can be achieved with e.g. a QR decomposition. Afterwards, the s_i, $i = 1, \ldots, 2^n - 1$ target vectors (where n is the dimension of the lattice) are generated according to Equation 3.

$$\mathcal{TV}(M) = \left\{ s = zM : z \in \{0, 1/2\}^n - \{0\} \right\} \tag{3}$$

After this step, the algorithm enters the main loop, composed of iterations that decode a target vector.

The decode procedure outputs $\mathcal{U}$, which is then processed according to Equation 4, resulting in set $\mathcal{X}$.

$$\begin{aligned} \gamma &= \min \left\{ ||\mathbf{uM} - \mathbf{s}|| \text{ for all } \mathbf{u} \in \mathcal{U} \right\} \\ \mathcal{X} &= \left\{ uM : u \in \mathcal{U}, ||uM - s|| = \gamma \right\} \end{aligned} \tag{4}$$

If set $\mathcal{X}$ has two, and only two vectors, then the solution is valid, a relevant vector has been found, and it is added to the list of relevant vectors $\mathcal{N}$. In this case, these vectors are symmetric to one another and, therefore, have the same norm (if the SVP is to be calculated, we only need to keep one of these vectors). Conversely, if only one or more than two vectors are returned in matrix $\mathcal{X}$, the result is *not* valid, and is discarded.

B. VORONOI 2.0

In [38], we proposed a number of modifications to this algorithm, resulting in an algorithm we call Voronoi 2.0, which we describe very briefly in this section, as we generalize our results for this algorithm as well (although this algorithm cannot be ported to an heterogeneous platform the same way as we do in this paper, as we will explain later on). In essence, we were able to 1) find a correlation between the target vectors and their solution (i.e. the solution of the decode procedure on them), concluding that, in general, the smaller the target vectors, the smaller their solution and 2) determine that, if target vectors are sorted by increasing norm, we only need to decode a given percentage (which varies depending on the used lattice basis) of the target vectors. This enabled us to implement improvements to specifically address the SVP, including:

- **Pruning**. We implement simple pruning (where we simply discard target vectors whose norm is larger than the norm of the shortest solution vector found up until some point), Gaussian pruning (where we also discard vectors, but according to the Gaussian heuristic, a popular heuristic in the context of SVP-solvers) and the previous two combined.
- **Sorting**. We determined that sorting the target vectors upfront would result in a speedup if pruning is used.

These optimizations yield a speed up relatively to Voronoi 1.0 that can be as high as 800x, as demonstrated in [38]. Additionally, in the same paper, we presented a CPU implementation that scales linearly with the number of CPU cores and a GPU implementation, which we briefly reproduce in Section IV, that is competitive with the CPU version.

IV. A CUDA GPU IMPLEMENTATION

This work is based on the first Voronoi-cell based algorithm proposed in the context of lattice-based cryptanalysis, by Agrell et al. [2]. In particular, this builds up on ideas that we used to implement a parallel GPU version of the algorithm [38], whose performance we show in Figure 2.

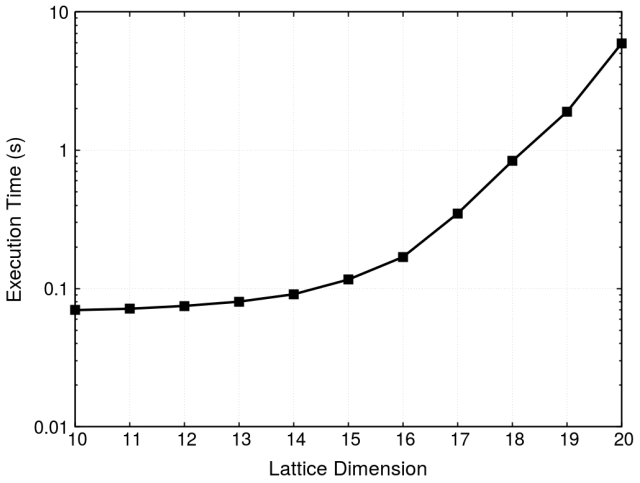

FIGURE 2. CUDA execution time (Machine A).

As with any other attack against lattice-based cryptosystems, the required time to solution grows exponentially. Although this algorithm is among the best from a complexity standpoint, it hits a memory wall pretty quickly (that is, for relatively low dimensions). This depends on how exactly the implementation is devised, but as a reference, we can safely say that if a thread decodes a single target vector, memory usage for dimension 25 would be 386 GB, ballpark. The more vectors are decoded per thread, the lower this figure, but the lower the parallelism degree in the GPU and therefore the lower the performance. In other words, for the implementation to be efficient, the memory requirements are too high and memory limitation is hit quickly.

On the other hand, there are no memory walls hit for sieving (and enumeration, more prominently) until at the very least dimension 80 or so e.g. [28], [30]. This was our motivation to design Voronoi 2.0; by pruning the set of target vectors it reduces both memory and computation requirements. Sadly, implementing Voronoi 2.0 on GPUs (and heterogenerous platforms, for that matter) has, as we will show, 2 major obstacles. The first problem concerns the use of memory. The memory available in one GPU is too small to test the algorithm on ''relatively interesting'' dimensions. Although Voronoi 2.0 is much less memory-demanding than the original Voronoi-cell algorithm, there are still considerable memory requirements. On a relatively modern and memory-capable GPU with, say, 6GB, we would probably not surpass dimension 21-23 with Voronoi 2.0 for safe pruning parameters (to understand the goal of pruning parameters, we refer the reader to [38]).

The second obstacle pertains to the impossibility of implementing (efficient) critical sections on GPUs (and heterogeneous systems). Our Voronoi 2.0 algorithm relies on critical sections in order to prune the set of target vectors to consider, a procedure that is repeated many times throughout the application. As such, we see this as a critical problem, which has no quick fix, and therefore we have started by studying the effect of the original Voronoi-cell based algorithm on heterogeneous systems first, which is the central goal of this paper. We believe that it is possible to infer the performance of Voronoi 2.0 on these systems, based on the results we present if this problem is solved. In fact, we comment on this later on in this paper.

V. WHY TO USE CPUS AND GPUS?

There are essentially 3 reasons why it makes sense to study Voronoi (both the original as well as Voronoi 2.0) on CPU+GPU computing platforms.

The first reason is related with security. Making use of all resources available on the computing platform is a critical problem in the context of lattice-based cryptanalysis, as we briefly explained in Section I and explain in more detail in Section VII, later in this paper. In short, this is because adversaries may have these computational resources at their disposal, and therefore it is critical for cryptographers to know how powerful a given attack is on such computing platforms. To the best of our knowledge, there are no published works on heterogeneous CPU+GPU attacks against lattice-based cryptosystems, although this has been done in the context of current, pre-quantum cryptosystems and their cryptanalysis e.g. [16], [36].

Another critical factor to consider is the power required to successfully attack a cryptosystem. In fact, some players may define key sizes based on the financial capability of possible adversaries, under the premise that certain attacks are not financially viable regarding the power they require to break a cryptosystem, which is intrinsically given by a dollar-day metric [40]. Since GPUs are known to be more power efficient than CPUs in terms of dollars per FLOP (= dollars per watt times watt per FLOP), even though this depends on the exact used GPU [1], it is fundamental to include them while assessing attacks in practice. In fact, it has been shown that CPU+GPU platforms can be even more energy/resource efficient than using CPUs or GPUs isolated [14], [24]. This aligns well with the natural energy consumption wall that needs to be investigated in the context of lattice-based cryptanalysis, which should be explored with the usage of heterogeneous systems, as we do in this paper.

The third reason is the natural memory increase, as we use more computing devices. Obviously, adding more computing devices increases the memory available and as we explained in Section IV, Voronoi is a memory-eager algorithm. This is also true for other SVP-attacks, such as sieving algorithms [26], [29], [30], which also hit a memory wall but also are able to trade memory consumption and execution time. This allows us to reach higher dimensions.

These reasons motivate an heterogeneous implementation, based on CPUs + GPUs, as detailed in the following.

VI. AN HETEROGENEOUS IMPLEMENTATION OF THE VORONOI CELL ALGORITHM

The core contribution of this work is the implementation of the Voronoi cell based algorithm on an heterogeneous platform, meaning an implementation that could make use of

the CPU and the GPU simultaneously. As mentioned before, such an implementation of Voronoi 2.0 has a major obstacle. This version of the algorithm has to use a critical section (or similar, as long as it is safe to use concurrent accesses) to update the minimum norm at a given moment. Sadly, neither CUDA nor StarPU allow critical sections among different SMs without compromising parallelization gains, a crucial feature for this implementation. There are software-based solutions that allow critical sections between SMs, but these are very inefficient since these operations must be executed sequentially by the CUDA threads (especially if if used often throughtout the application, as in this case) and would render our implementation too slow. As a critical section is not feasible within the GPU, we were forced to implement the original Voronoi cell algorithm on our CPU+GPU setup, pushing the problem of concurrent updates of the minimum norm to future work. Yet, we can learn from this and extrapolate on a few conclusions if Voronoi 2.0 would be implemented, as we will show later in this paper.

A. THE STARPU FRAMEWORK IN A NUTSHELL

We have implemented an heterogeneous implementation on top of the StarPU framework [6]. This framework integrates the concepts of ''codelets'' and ''tasks''. StarPU defines a codelet as ''a computational kernel that can possibly be implemented on multiple architectures such as a CPU, a CUDA device or an OpenCL device'', while a task ''consists in applying a codelet on a data set, on one of the architectures on which the codelet is implemented''. Tasks in StarPU, as is the case with CUDA kernels, are launched asynchronously. These tasks are executed by workers, which are processing units or parts of one.

B. CONSIDERATIONS ON OUR IMPLEMENTATION

There is a number of aspects to consider and optimize in a CPU+GPU implementation of the Voronoi algorithm, including 1) workload distribution, 2) scheduling of tasks and 3) memory management, which we address in the following.

1) WORKLOAD DISTRIBUTION

As derived from the StarPU framework, we can view our implementation as a set of tasks, which can run on a CPU core or a (whole) GPU. Each task consists of a given fixed number of target vector decode instances; the number of tasks depends on the lattice dimension, as it defines the number of decodes to perform. All in all, defining the workload distribution in this model boils down to choosing the number of tasks (the more tasks we create, the fewer decodes there are within each task).

The best granularity of tasks in terms of throughput performance can only be determined empirically. In the context of lattice-based cryptanalysis, this is a major flaw, because attacks are only ran once per lattice basis/input data; nevertheless, we did conduct tests to determine the best task granularity. Also, if we were to develop a CPU+GPU implementation by hand, this would be a trivial problem, as we could run a small part of the algorithm on a small data set on both units and thus infer a good workload distribution.

We have implemented codelets for both the CPU and GPU, the latter being based on CUDA. Each of these codelets allocate auxiliary memory for the target vectors to be decoded, and the memory is freed once the kernel is finished.

This creates some computing stress, especially on the GPU side, but there is no way, in our opinion, to mitigate this; we note, however, that this is not as problematic as actually transferring data over the PCI-e bus on every executed task, which does not happen in our implementation.

2) SCHEDULING

The StarPU framework incorporates several different schedulers, assigning tasks to the computing units (that is, the CPU cores and the GPU(s)) transparently to the user. We have used the eager scheduler, which stores tasks on a central task repository which workers (either a CPU core or a GPU card) take work (or tasks, in the introduced nomenclature) from. This mechanism does not include data prefetching, because the scheduler does not know upfront on what worker tasks will be executed.

A very relevant aspect of scheduling optimization is the granularity of tasks. We have experimented with this, running several different granularities and assessing the ultimate results of the StarPU framework. In our particular application, the granularity is actually given by the number of target vectors incorporated on each task. We have tested runs with tasks encapsulating a number of target vectors that can range from 2^{10} to 2^{16} (the number of target vectors for each dimension is itself a power of two). To better understand how tasks are formed, we have drawn Figure 4 as well. These tests were done for 22 CPU threads and 2 GPU cards, our best computing setup (StarPU requires one CPU thread to manage each GPU). The results are shown in Figure 3, for 22 threads and 2 GPU cards and we also show the performance for 1-22 CPU threads and 2 GPUs cards on Table 2 (in the table, we excluded task granularity $= 2^{16}$ as it performs considerably worse than the other tested granularities, as shown in Figure 3).

Given that our StarPU implementation performs best for tasks entailing 2^{12} target vectors on the maximum input size tested, lattices in dimension 20, we have fixed this task granularity for all the results presented in the rest of this paper. As it is amortized over several runs, all results exclude the GPU wake-up time, i.e. the time it requires to ''activate'' the GPU or the time elapsed between launching the StarPU application and the GPU actually becoming available to execute work.

It is important to say that while the overall (that is CPU RAM and GPU RAM) memory of the system limits the maximum possible lattice dimension, the memory in each GPU card also poses a challenge.

That is well verified in our tests; up until dimension 19, the GPU kernel that processes a task will be launched with 2^x threads (for task granularity $= 2^y$). However, in dimension 20, due to the memory limitation on the GPUs (5 GBs),

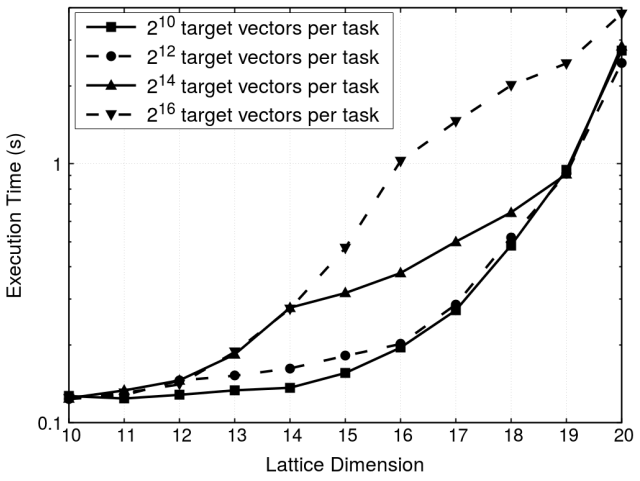

FIGURE 3. Performance of our application implemented on the StarPU framework for different task granularities (number of target vectors per task). All results were obtained running 22 threads and 2 GPU cards (machine B).

we are forced to launch a maximum of $2^x/2$ threads per GPU kernel, as each thread allocates additional memory per target vector decoded (in particular, more memory as the dimensions grows). This means that, for dimension 20, each thread running on the GPU will decode 2 target vectors rather than 1.

If we were able to run higher lattice dimensions (which we cannot as our system is limited to a total of 74 GBs, i.e., 64GBs of CPU RAM and 5GB per GPU), the number of GPU threads launched per task would continue to drop, as the memory required by each GPU thread would grow. Naturally, that this would impact the performance of the GPUs and therefore the overall performance of our StarPU application.

3) DATA MANAGEMENT

StarPU offers two functions to allocate data, starpu_malloc() and starpu_memory_pin(). Either way, memory is (possibly) transferred asynchronously, thus permitting data transfer to be overlapped with computation. Data management is entirely up to the framework.

In fact, this is not very relevant for our application as the only data that is actually transferred over the PCI-e bus is, at the end of the application, the set of relevant vectors decoded on the GPU side (which is actually partitioned among blocks, so that all tasks are data-independent before StarPU). This matrix is once again generated by aggregating all blocks at the end of the application, which is illustrated in Figure 4; we let StarPU manage data coherence. This is simpler than managing memory explicitly, especially with dynamic scheduling of tasks.

C. PERFORMANCE BENCHMARKS AND RESULTS

We have tested our implementation extensively to determine not only its overall performance (and actual gain in using GPUs) but also the overhead that is incurred by using StarPU.

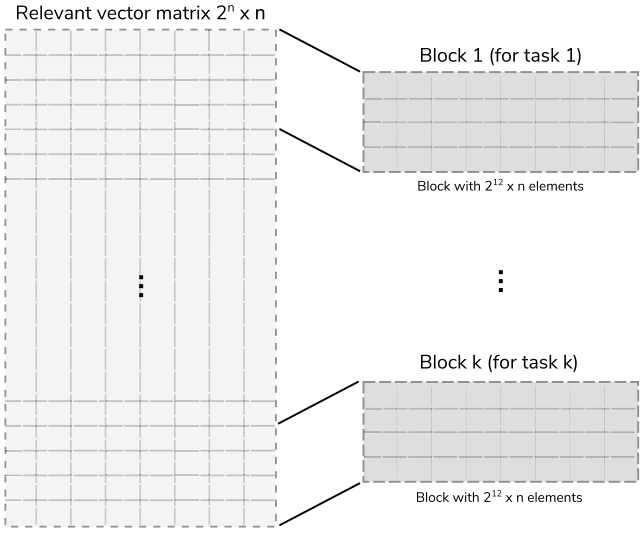

FIGURE 4. Partition of the matrix that holds the 2^n relevant vectors in blocks with the best performing granularity, 2^{12} target vectors per task. Note that each block has $2^{12} \times n$ positions as each vector has n coordinates.

TABLE 2. Performance of our application implemented on the StarPU framework for different task granularities (number of target vectors per task), for 1-22 CPU threads and 2 GPU cards (machine B), in seconds.

Thread count / Dimension	1	2	4	8	16	22
Dim. 18 2^{10}	2.121	1.726	1.292	0.884	0.580	0.484
Dim. 18 2^{12}	1.091	1.027	0.896	0.717	0.561	0.520
Dim. 18 2^{14}	1.169	1.163	0.906	0.662	0.667	0.649
Dim. 19 2^{10}	4.884	3.971	2.875	1.902	1.216	0.945
Dim. 19 2^{12}	2.379	2.186	1.838	1.478	1.047	0.915
Dim. 19 2^{14}	2.369	2.001	1.947	1.378	1.323	0.911
Dim. 20 2^{10}	22.53	16.31	10.51	6.219	3.535	2.733
Dim. 20 2^{12}	9.442	8.210	6.499	4.651	3.029	2.452
Dim. 20 2^{14}	6.909	6.277	5.494	3.954	2.934	2.849

1) STARPU'S OVERHEAD

StarPU's semi-automatic data scheduling and memory management does not come for free. We have tested the exact overhead introduced by StarPU so that it does not cloud the gains that we have by using multiple GPUs. To that end, we have implemented a parallel CPU-only version of the algorithm in standard C++, with OpenMP, and compared that against the StarPU version. We have tested multiple thread counts so that we can comment on the scalability both with and without StarPU. Figure 5 shows the performance of both versions, for 1-24 threads, and for readability purposes we also show the results for dimensions 18-10 in Table 3.

Looking at the figure, there is one result that stands out: the overhead is much more noticeable for lower dimensions. Note, however, that until dimension 17, most runs take less than 1 second and therefore these results should be taken

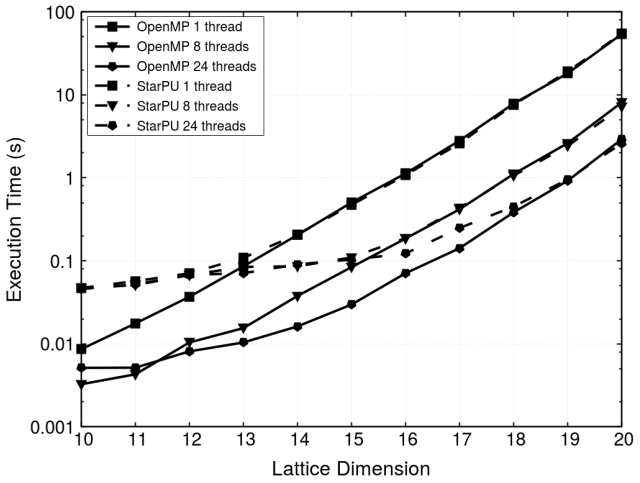

FIGURE 5. Performance of our application implemented on OpenMP and StarPU (without GPUs), for 1-24 threads (machine B).

TABLE 3. Performance of the CPU version (OpenMP vs StarPU), for 1, 2, 4, 8, 16 and 24 threads, in seconds. Results obtained on Machine B.

Thread Count	1	2	4	8	16	24
Dim. 18 OpenMP	7.831	4.565	2.248	1.111	0.558	0.384
Dim. 18 StarPU	7.593	3.960	1.965	1.063	0.595	0.451
Dim. 19 OpenMP	18.09	11.27	5.305	2.610	1.311	0.922
Dim. 19 StarPU	19.02	9.392	4.535	2.417	1.295	0.955
Dim. 20 OpenMP	53.82	32.58	15.89	8.074	4.042	2.918
Dim. 20 StarPU	54.58	28.16	13.81	7.142	3.708	2.598

with a pinch of salt. As we want to run the algorithm with as many threads as possible, the overhead introduced by StarPU on the CPU side is negligible (and sometimes, StarPU is even faster). Although we are not sure why this happens, it could be that the scheduler of StarPU is more efficient than the OpenMP dynamic scheduler, the scheduler we use. Another possible test is the overhead that StarPU introduces on the CPU side when using GPUs. This could be tested using the OpenMP and CUDA versions for the same workload that StarPU ultimately assigns to the CPU and the GPU, respectively. However, it should come at no surprise that StarPU's memory management and scheduling mechanisms incur potentially large overheads and there is not much to be learned from that, in our view.

As the lower dimensions are executed too quickly and all the results were obtained with Turbo Boost technology turned on, showing the actual scalability figures is not particularly relevant. However, we can say that our StarPU application's scalability is in line with our OpenMP implementaiton, which we have shown to scale linearly in previous work.

2) OVERALL PERFORMANCE

The most important test to be made is the actual overall performance of our application simultaneously running on

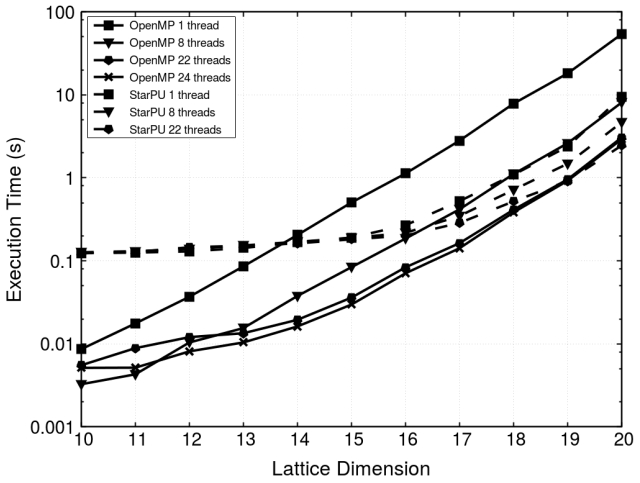

FIGURE 6. Performance of our OpenMP implementation (1-24 threads) and our StarPU implementation (1-22 CPU threads and 2 GPU cards). Results obtained on machine B.

TABLE 4. Performance of our OpenMP implementation (1-24 threads) and our StarPU implementation (1-22 CPU threads and 2 GPU cards). (NP = Not Possible) Results obtained on Machine B.

Thread Count	1	2	4	8	16	22	24
Dim. 18 OpenMP	7.831	4.565	2.248	1.111	0.558	0.413	0.384
Dim. 18 StarPU	1.091	1.027	0.896	0.717	0.561	0.520	NP
Dim. 19 OpenMP	18.09	11.27	5.305	2.610	1.311	0.958	0.922
Dim. 19 StarPU	2.379	2.186	1.838	1.478	1.047	0.915	NP
Dim. 20 OpenMP	53.82	32.58	15.89	8.074	4.042	3.056	2.918
Dim. 20 StarPU	9.442	8.210	6.499	4.651	3.029	2.452	NP

CPUs and GPUs, which we show in Figure 6. For readability purposes we also show the results for dimensions 18-20 in Table 4.

The results deserve a few comments. First, both implementations scale well, although as shown in Table 4, this can only be seen in dimension 20, because other dimensions do not run long enough for this to show up. Secondly, our heterogeneous implementation (on StarPU) is better than the OpenMP version if we are to compare StarPU running 22 threads and 2 graphic cards and OpenMP with 24 threads (as mentioned, StarPU controls the execution of each GPU with a CPU thread, thus only allowing 22 threads for actual work). This is already true for dimension 19, even though the execution time is very small for the higher thread counts. In dimension 20, where the application takes about 3 seconds with 24 OpenMP threads, we obtain a gain of roughly 20% with our heterogeneous implementation. We expected this gap to grow with the lattice dimension, should higher dimensions be possible to run on this computing system. For higher dimensions, the penalty paid for transferring memory over the PCIe bus will dilute (with 22 threads and 2 GPUs, the execution takes less than 3 seconds and therefore speedups are not as noticable as if the execution took for, say, 30 minutes).

The gain obtained by our heterogeneous implementation (or, conversely, the use of GPUs) is more noticeable for 1 thread, as GPU's represent a proportionally higher computing capability in this scenario. This can be observed in the first column for the thread counts, which compares the OpenMP implementation with a single thread versus StarPU with 1 CPU thread and 2 graphic cards. In this setup, our speedup reaches 7.18x, 7.6x and 5.7x for dimension 18, 19 and 20, respectively.

We can observe that the speedup decreases with the thread count, because the added computing power with the 2 graphic cards becomes proportionally lower as we increase the number of threads (a good example to understand this is that if we had, say, 1000 threads, adding these 2 GPU cards would not make much - or any - difference). We believe that the overall speedup is smaller for dimension 20, compared to the lower dimensions, because the GPU threads decode 2 target vectors instead of 1, a problem we have mentioned before. Dimension 20 is precisely the maximum input size that saturates the memory on the GPUs side. Of course that, in theory, we could use Direct Memory Access (DMA) to overcome this problem, but the overall memory of the system would hit a wall anyways for a relatively low lattice dimension (e.g. dimension 22). Plus, DMA would increase memory transfer latency considerably, which would not be interesting in this particular case. The central goal of this paper is to prove that heterogeneous computing systems can be used successfully to attack lattice-based cryptossytems.

We also conducted benchmarks to determine the workload of each device. The GPUs executed 25% of the overall workload (32 out of 128 StarPU-tasks), for dimension 19, and 17% (44 out of 256) for dimension 20. We also measured the average idle time of the GPUs (only for seed 0), which came out at 131ms (i.e. 14% of the total execution time) for dimension 19 and 159ms, for dimension 20 (i.e. 6.5% of the total execution time). We believe that if it was possible to test higher dimensions, the idle time of the GPUs would continue to decrease as more workload gives the scheduler a chance to optimize distribution.

Another interesting test would be to determine the gains of a single GPU, however this is not possible for the highest dimensions tested due to lack of memory. In fact, this is a good example of the arguments presented in Section V, where we argued that an heterogeneous CPU+GPU implementation would make sense as it increases the overall memory available, therefore allowing for higher dimensions.

A short note on inferring results for heterogeneous Voronoi 2.0 without actual testing: we believe that Voronoi 2.0 would also be accelerated, with good results, on such an environment, even if critical sections were implemented via software. This is because its computational essence is identical to that of the original Voronoi algorithm, and the only difference between the algorithms is that Voronoi 2.0 discards some computation that Voronoi actually performs.

The efficiency of our heterogenous implementation can also be assessed on the CPU and the GPU level. At this level two distinct scenarios occur: 1) the CPU exploits the large amounts of cache memory available and the ability to deal efficiently with divergence, namely through branch predictors, execution out of order mechanisms, etc., since control flow exists in the Voronoi algorithm when decoding target vectors, but it looses in terms of compute capability due to the limited number of cores available; 2) the GPU, on the other hand, exploits quite well data-parallelism due to the thousands of cores available, which is a known characteristic of the Voronoi algorithm in parallel, as several decodes could be performed in parallel and several coordinates of vectors could be assessed in parallel, although it contains a smaller cache memory and performs a significant number of slow global memory accesses, that also originate bank conflicts, retrieving/sending data from/to DRAM (please recall that massive amounts of memory are necessary for running the algorithm).

VII. PRACTICAL IMPACT OF OUR CONTRIBUTIONS AND RELATED WORK

As mentioned before, this is the first ever heterogeneous CPU+GPU implementation of a SVP-attack against lattice-based cryptosystems.

A. RELATION WITH CRYPTOGRAPHY

Security parameters for these systems are usually considered based on the extrapolation of implementations of attacks in practice. For instance, if we determine that an attack is feasible using 32 CPU cores with a given security parameter, then we choose a parameter such that the attack becomes intractable on that computing platform. However, if one argues that a given algorithm can only make use of a specific computing platform (e.g. CPUs), then extrapolations disregard the use of other platforms or cumulative hardware resources, such as GPUs. This opens up a gap in security, as adversaries may be able to use these computing platforms and therefore come up with attacks that are practical, even for strong security parameters chosen based on the performance of other, weaker computing platforms. This is why it is so important to determine the performance of attacks in practice, on the best possible computing platforms.

For instance, we have done curve fitting on the performance of our implementations, and the execution time of our OpenMP CPU implementation grows according to $4.57 \times 10^{-8}\exp(0.8852n)$, the GPU CUDA implementation seems to grow according to $4.489 \times 10^{-7}\exp(0.8027n)$, and our heterogeneous implementation, with 22 threads and 2 GPUs, only grows according to $3.652 \times 10^{-7}\exp(0.7847n)$. For visual purposes, we plot these curves in Figure 7 for relevant values of n, the dimension of the lattice. For dimension 40, the OpenMP version would take 3.46 years, our heterogeneous implementation on StarPU would take 181 days and the CUDA version would take 1.25 years. This can make a huge difference when determining security parameters. Although cryptographers define security parameters such as the key size, let us think about.

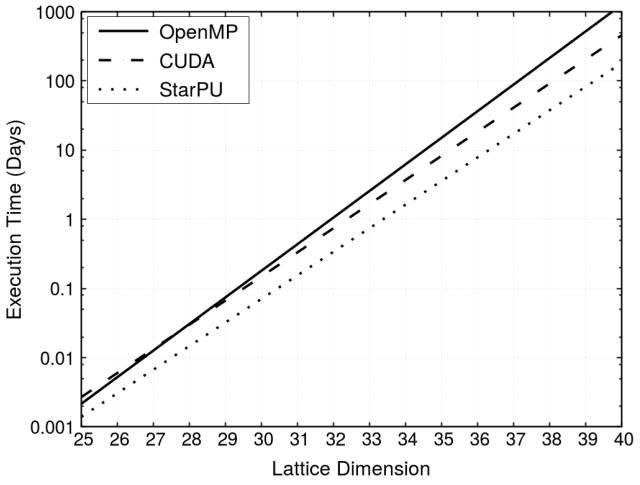

FIGURE 7. Extrapolation of the execution times of Voronoi on a CPU, on a GPU and on the CPU+GPU platforms, based on curve fitting.

B. RELATED WORK

There are, to our knowledge, two papers exploring CPUs and GPUs in the context of lattice-based cryptanalysis. The first paper, published in 2011, introduced several important concepts such as a new bounding function for enumeration with pruning, and the price of breaking a system, in dollars, depending on a few variables. The paper also presented an SVP-attack (enumeration with extreme pruning) on CPUs and GPUs, but in a different manner than our attack: the authors randomize several lattice basis and run several instances of BKZ (an advanced lattice-basis reduction algorithm) and enumeration on them. Therefore, this explores course-grained parallelism, running several instances of the same algorithm on different bases. The number of instances started depends on the success probability of each instance. This means that if the number of instances is low, then the parallelism is also low. Plus, this is not generalizable to other attcks (e.g. Voronoi and Sieving), because these are based on a single instance of the algorithm running on a single basis.

The second paper, published in 2014, implements a relaxed version of LLL, called ''cost-reduced MB-LLL'' or CR-MB-LLL, a lattice reduction algorithm, on heterogeneous systems [23]. The authors showed that the heterogeneous implementation of the CR-MB-LLL algorithm performs better than the multi-threaded version for CPUs, if large matrices (i.e. lattice bases) are provided as input. It is important to say that this cannot be used as an SVP-attack, but rather a α-SVP attack (a relaxed version of the SVP where the solution is at most α% off the optimal solution) or a lattice reduction algorithm, which is typically applied before a ''hard'' SVP-attack.

We hope that our paper fosters the community to explore heterogeneous systems with other algorithms (e.g. HashSieve and LDSieve [7], [26] and, recently, the General Sieve Kernel [5]), which have been used to break higher dimensions of the SVP-challenge with CPUs alone.

VIII. CONCLUSION

We have shown, for the first time, that heterogeneous CPU+GPU hardware platforms can be used to attack lattice-based cryptosystems. Although possible in theory, this had never been shown with any algorithm before, to our best knowledge. This opens a new realm of possibilities, such as determining security parameters for cryptosystems more assertively, as we now should consider platforms with multiple GPUs in such extrapolation.

In particular, we have shown that Voronoi cell-based algorithms, important algorithms in the context of lattice-based cryptanalysis are suited for these platforms. Our results show that using two not-so-modern GPUs (released in Jan. 2013) in addition to a 24-core CPU system delivered a maximum 7.6x speedup, when compared to the performance of the CPUs alone. In fact, this remained true even considering that StarPU, the framework we implemented the algorithm on, has to use a CPU-core to manage each additional GPU, thus turning them off for computation; in other words, 22 CPU-cores with 2 GPUs was a better setup for our application than 24 CPU-cores.

Voronoi's algorithm conditional statements limit the performance that can be extracted from the GPUs due to threads divergence.

Future Work. We plan to investigate Voronoi 2.0 - an algorithm we devised and proposed on another paper - on CPU+GPUs in the future. To this end, we may consider making our implementation DMA-capable to eliminate memory walls on the GPU(s) side. We are currently investigating algorithm optimizations which results on lower thread divergence on the GPUs; additionally, future experiments will be executed on NVidia GPUs with either Volta or Turing architectures - these support independent thread scheduling, which minimizes the impact of thread divergence and optimizes synchronization primitives [37], therefore potentially allowing for the implementation of Voronoi 2.0.

REFERENCES

[1] Y. Abe, H. Sasaki, S. Kato, K. Inoue, M. Edahiro, and M. Peres, ''Power and performance of GPU-accelerated systems: A closer look,'' in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, Sep. 2013, pp. 109–110.

[2] E. Agrell, T. Eriksson, A. Vardy, and K. Zeger, ''Closest point search in lattices,'' *IEEE Trans. Inf. Theory*, vol. 48, no. 8, pp. 2201–2214, Aug. 2002.

[3] M. Ajtai, R. Kumar, and D. Sivakumar, ''A sieve algorithm for the shortest lattice vector problem,'' in *Proc. 33rd Annu. ACM Symp. Theory Comput.*, 2001, pp. 601–610.

[4] S. Akleylek, N. Bindel, J. Buchmann, J. Krämer, and G. A. Marson, ''An efficient lattice-based signature scheme with provably secure instantiation,'' in *Proc. 8th Int. Conf. Prog. Cryptol. (AFRICACRYPT)*, vol. 9646. Berlin, Germany: Springer-Verlag, Apr. 2016, pp. 44–60. [Online]. Available: https://eprint.iacr.org/2016/030. doi: 10.1007/978-3-319-31517-1_3.

[5] M. R. Albrecht, L. Ducas, G. Herold, E. Kirshanova, E. W. Postlethwaite, and M. Stevens ''The general sieve kernel and new records in lattice reduction,'' Cryptol. ePrint Arch., Tech. Rep. 2019/089, 2019. [Online]. Available: https://eprint.iacr.org/2019/089

[6] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier, ''StarPU: A unified platform for task scheduling on heterogeneous multicore architectures,'' in *Euro-Par 2009 Parallel Processing*, H. Sips, D. Epema, and H.-X. Lin, Eds. Berlin, Germany: Springer, 2009, pp. 863–874.

[7] A. Becker, L. Ducas, N. Gama, and T. Laarhoven, "New directions in nearest neighbor searching with applications to lattice sieving," in *Proc. 27th Annu. ACM-SIAM Symp. Discrete Algorithms (SODA)*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2016, pp. 10–24. [Online]. Available: http://dl.acm.org/citation.cfm?id=2884435.2884437
[8] D. Bernstein, J. Buchmann, and E. Dahmen, Eds., *Post-Quantum Cryptography*. Berlin, Germany: Springer-Verlag, 2009. [Online]. Available: http://www.springerlink.com/content/978-3-540-88701-0
[9] J. W. Bos, M. Naehrig, and J. van de Pol, "Sieving for shortest vectors in ideal lattices: A practical perspective," Cryptol. ePrint Arch., Tech. Rep. 2014/880, 2014. [Online]. Available: https://eprint.iacr.org/2014/880
[10] J. Buchmann, E. Dahmen, and A. Hülsing, "XMSS—A practical forward secure signature scheme based on minimal security assumptions," in *Proc. 4th Int. Conf. Post-Quantum Cryptogr. (PQCrypto)*, 2011, pp. 117–129.
[11] Y. Chen and P. Q. Nguyen, "BKZ 2.0: Better lattice security estimates," in *Proc. Adv. Cryptol.-ASIACRYPT, 17th Int. Conf. Theory Appl. Cryptol. Inf. Secur.* Berlin, Germany: Springer, 2011, pp. 1–20.
[12] F. J. G. Correia, "Assessing the hardness of SVP algorithms in the presence of CPUs and GPUs," M.S. thesis, Dept. Inform., Univ. Minho, Braga, Portugal, 2014.
[13] O. Dagdelen and M. Schneider, "Parallel enumeration of shortest lattice vectors," in *Proc. Euro-Par-Parallel Process., 16th Eur. Conf. Parallel Process.*. Berlin, Germany: Springer, 2010, pp. 211–222.
[14] T. Dong, V. Dobrev, T. Kolev, R. Rieben, S. Tomov, and J. Dongarra, "A step towards energy efficient computing: Redesigning a hydrodynamic application on CPU-GPU," in *Proc. IEEE 28th Int. Parallel Distrib. Process. Symp.*, May 2014, pp. 972–981.
[15] L. Ducas, A. Durmus, T. Lepoint, and V. Lyubashevsky, "Lattice signatures and bimodal Gaussians," *IACR Cryptol. ePrint Arch.*, vol. 2013, p. 383, 2013.
[16] H. M. Fadhil and M. I. Younis, "Parallelizing RSA algorithm on multicore CPU and GPU," *Int. J. Comput. Appl.*, vol. 87, pp. 15–22, Feb. 2014.
[17] N. Gama, P. Q. Nguyen, and O. Regev, "Lattice enumeration using extreme pruning," in *Proc. EUROCRYPT*, 2010, pp. 257–278.
[18] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Dept. Comput. Sci., Stanford, CA, USA, 2009, Art. no. aAI3382729.
[19] O. Goldreich, S. Goldwasser, and S. Halevi, "Public-key cryptosystems from lattice reduction problems," in *Proc. 17th Annu. Int. Cryptol. Conf. Adv. Cryptol. (CRYPTO)*. London, U.K.: Springer-Verlag, 1997, pp. 112–131. [Online]. Available: http://dl.acm.org/citation.cfm?id=646762.706185
[20] J. Hoffstein, J. Pipher, and J. H. Silverman, "NTRU: A ring-based public key cryptosystem," in *Algorithmic Number Theory* (Lecture Notes in Computer Science). Berlin, Germany: Springer-Verlag, 1998, pp. 267–288.
[21] T. Ishiguro, S. Kiyomoto, Y. Miyake, and T. Takagi, "Parallel Gauss Sieve algorithm: Solving the SVP challenge over a 128-dimensional ideal lattice," in *Public-Key Cryptography—PKC* (Lecture Notes in Computer Science), vol. 8383. Berlin, Germany: Springer, 2014, pp. 411–428.
[22] J. Ding, X. Xie, and X. Lin, "A simple provably secure key exchange scheme based on the learning with errors problem," Cryptol. ePrint Arch., Tech. Rep. 2012/688, 2012. [Online]. Available: https://eprint.iacr.org/2012/688
[23] C. M. Józsa, F. Domene, A. M. Vidal, G. Piñero, and A. González, "High performance lattice reduction on heterogeneous computing platform," *J. Supercomput.*, vol. 70, no. 2, pp. 772–785, Nov. 2014. doi: 10.1007/s11227-014-1201-2.
[24] K. Kothapalli, D. S. Banerjee, P. J. Narayanan, S. Sood, A. K. Bahl, S. Sharma, S. Lad, K. K. Singh, K. Matam, S. Bharadwaj, R. Nigam, P. Sakurikar, A. Deshpande, I. Misra, S. Choudhary, and S. Gupta, "CPU and/or GPU: Revisiting the GPU vs. CPU myth," *CoRR*, vol. abs/1303.2171, Mar. 2013, pp. 1–20.
[25] 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 *Proc. Cryptograph. Hardw. Embedded Syst.-CHES, 13th Int. Workshop Cryptograph. Hardw. Embedded Syst.* Berlin, Germany: Springer, 2011, pp. 176–191.
[26] T. Laarhoven, "Sieving for shortest vectors in lattices using angular locality-sensitive hashing," in *Proc. CRYPTO*, 2015, pp. 3–22.
[27] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász, "Factoring polynomials with rational coefficients," *Math. Annabel*, vol. 261, no. 4, pp. 515–534, 1982.
[28] A. Mariano, T. Laarhoven, and C. Bischof, "A parallel variant of LDSieve for the SVP on lattices," in *Proc. 25th Euromicro Int. Conf. Parallel, Distrib. Netw.-Based Process. (PDP)*, Mar. 2017, pp. 23–30.
[29] A. Mariano, "High performance algorithms for lattice-based cryptanalysis," Ph.D. dissertation, Dept. Comput. Sci., Technische Universität Darmstadt, Darmstadt, Germany, 2016.
[30] A. Mariano and C. Bischof, "Enhancing the scalability and memory usage of HashSieve on multi-core CPUs," in *Proc. PDP*, 2016, pp. 545–552.
[31] A. Mariano, T. Laarhoven, F. Correia, M. Rodrigues, and G. Falcão, "A practical view of the state-of-the-art of lattice-based cryptanalysis," *IEEE Access*, vol. 5, pp. 24184–24202, 2017.
[32] P. Martins, L. Sousa, and A. Mariano, "A survey on fully homomorphic encryption: An engineering perspective," *ACM Comput. Surv.*, vol. 50, no. 6, p. 83, 2017.
[33] D. Micciancio and O. Regev, "Lattice-based cryptography," in *Post-Quantum Cryptography*. Berlin, Germany: Springer, 2009, pp. 147–191.
[34] D. Micciancio and P. Voulgaris, "A deterministic single exponential time algorithm for most lattice problems based on Voronoi cell computations," in *Proc. STOC*, 2010, pp. 351–358.
[35] P. Q. Nguyen and J. Stern, "The two faces of lattices in cryptology," in *Proc. CaLC*, 2001, pp. 146–180.
[36] E. Niewiadomska-Szynkiewicz, M. Marks, J. Jantura, M. Podbielski, and P. Strzelczyk, "Comparative study of massively parallel cryptalysis and cryptography on CPU-GPU cluster," in *Proc. Mil. Commun. Inf. Syst. Conf.*, Oct. 2013, pp. 1–8.
[37] NVIDIA. (2017). *Nvidia Tesla V100 GPU Architecture*. [Online]. Available: https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf
[38] *Omitted for Blind Review*, Omitted, 2019.
[39] O. Regev, *Lattice-Based Cryptography*. Berlin, Germany: Springer, 2006, pp. 131–141.
[40] M. Rückert and M. Schneider, "Estimating the security of lattice-based cryptosystems," *IACR Cryptol. ePrint Archive*, vol. 2010, p. 137, Jan. 2010.
[41] C. P. Schnorr, "A hierarchy of polynomial time lattice basis reduction algorithms," *Theor. Comput. Sci.*, vol. 53, nos. 2–3, pp. 201–224, 1987.
[42] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Proc. 35th Annu. Symp. Found. Comput. Sci. (SFCS)*. Washington, DC, USA: IEEE Computer Society, Nov. 1994, pp. 124–134. doi: 10.1109/SFCS.1994.365700.
[43] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM J. Comput.*, vol. 26, no. 5, pp. 1484–1509, 1997. doi: 10.1137/S0097539795293172.

GABRIEL FALCAO (S'07–M'10–SM'14) received the Ph.D. degree from the University of Coimbra, in 2010, where he is currently an Assistant Professor with the Department of Electrical and Computer Engineering. In 2011 and 2017, he was a Visiting Professor with EPFL, and in 2018, he was a Visiting Academician with ETHZ, Switzerland. He is also a Researcher with the Instituto de Telecomunicações. His research interests include parallel computer architectures, energy-efficient processing, GPU- and FPGA-based accelerators, and compute-intensive signal processing applications. He is a member of the IEEE Signal Processing Society and a Full Member of the HiPEAC Network of Excellence.

FILIPE CABELEIRA received the B.Sc. and M.Sc. degrees in electrical and computer engineering from the Faculty of Science and Technology, University of Coimbra, Portugal, in 2017 and 2019, respectively. His work focuses on the implementation of efficient parallel algorithms for lattice-based cryptography on multicore CPUs and GPUs. His research interests include parallel computing and lattice-based cryptography.

ARTUR MARIANO received the Ph.D. degree from the Darmstadt University of Technology, Germany. He holds a postdoctoral position with CSIG, INESC TEC, University of Minho, Portugal. He works on lattice-based cryptanalysis, with particular focus on understanding the efficiency of attacks on modern and high-end computer architectures. He is currently involved with multi-disciplinary national and international projects aiming at understanding and improving the practicability of lattice-based cryptanalysis. He was a recipient of the DFG Postdoctoral Grant to work on high-performance lattice-based cryptography, is currently hosted by Luis Paulo Santos, and is looking to establishing his own research group.

LUIS PAULO SANTOS is currently an Assistant Professor with the Department of Informatics, Universidade do Minho, Portugal. He lectures computer architecture and computer graphics. In 2019, he joined the International Iberian Nanotechnology Laboratory, Quantum Software Engineering Group, Braga, Portugal, as a Research Associate. He has spent several periods as an Invited Researcher on a few international institutions, such as the University of Bristol, U.K., the Warwick Manufacturing Group, University of Warwick, U.K., Université de Rennes I, France, and the Texas Advanced Computing Center, University of Texas at Austin, USA. He has been the Vice-Director of the Department of Informatics and of its bachelor's and M.Sc. degrees in computer science. He has been the Director of the Doctoral Program on Informatics. He is a Senior Researcher of INESC TEC, Portugal. More recently, he became interested on quantum computing and its applications to global illumination, and graphics and numerical integration, in general. His research interests include global illumination and parallel processing.

Dr. Santos has been the Chair of the Eurographics Portuguese Chapter, since 2016. He has published several articles on conferences and journals within these areas of knowledge. He participated on several research projects, supervised five Ph.D. students, and acted as an Associate Editor of Elsevier's *Computers and Graphics*, from 2011 to 2019.