Fast GPU Implementation of Dumer’s Algorithm
Solving the Syndrome Decoding Problem

Shintaro Narisada, Kazuhide Fukushima and Shinsaku Kiyomoto
KDDI Research, Inc.
Fujimino, Japan
{sh-narisada,ka-fukushima,kiyomoto}@kddi-research.jp

Abstract—The syndrome decoding problem (SDP) is a security basis for code-based cryptographic algorithms such as the McEliece cryptosystem, which has been selected as a finalist for the third round of the post-quantum cryptography (PQC) standardization project organized by the National Institute of Standards and Technology (NIST). Information set decoding (ISD) is a general term for algorithms that efficiently solve SDPs based on combinatorial enumeration. In this paper, we focused on Dumer’s algorithm, which is relatively lightweight among the ISD algorithms; and proposed a multi-parallel Dumer’s algorithm optimized for computing devices that perform massively parallel processing. We implemented our algorithm on a GPU and tested it by solving some SDP instances in the decoding challenge hosted by Inria. We report that our algorithm first solved a 1161-dimensional SDP in the Goppa-McEliece setting in approximately 380 hours, which has a computational complexity of $2^{325}$. Our results contribute to a more detailed security analysis for code-based cryptosystems.

Index Terms—Syndrome Decoding, Information Set Decoding, Graphics Processing Unit (GPU)

I. INTRODUCTION

Code-based cryptography is a cryptographic scheme based on so-called error correcting code. The first code-based cryptography was the McEliece cryptosystem [1] proposed in 1978. Although the McEliece cryptosystem is an old cryptosystem proposed approximately half a century ago, it still attracts attention as a quantum-resistant cryptosystem and is currently a finalist candidate in the NIST PQC standardization process. The syndrome decoding problem (SDP) is an NP-complete problem that is a fundamental assumption for the security of code-based cryptosystems. Exponential time algorithms to solve the SDP efficiently have been previously studied. Prange [2] constructed an algorithm to efficiently search for the solution to the SDP by decomposing the input matrix into an identity matrix and random noise by applying elementary row operations. The Lee and Brickell algorithm proposed by Lee et al. [3] generalized Prange algorithm and reduced the time complexity of Prange algorithm. Stern [4] and Dumer [5] succeeded in further reducing the time complexity of solving the SDP by applying a matching algorithm called birthday decoding to the Lee and Brickell algorithm. In 2011, Bernstein et al. [6] proposed an algorithm referred to as the ball-collision decoding algorithm that allows partial matching in the matching phase, unlike the birthday decoding in Dumer’s algorithm. Ball-collision decoding is theoretically faster than Dumer’s algorithm. May et al. [7] and Becker et al. [8] improved Dumer’s algorithm by using the divide-and-conquer algorithm, which is often used to solve subset sum problems. May et al. [9] provided an algorithm applying the nearest neighbor search to Dumer’s algorithm that is faster at solving the SDP than Becker’s algorithm, which was later optimized by Both and May [10], [11]. These algorithms for solving the SDP, including Prange’s algorithm, are collectively called information set decoding (ISD) algorithms.

A fast implementation of the ISD algorithm is important for a more rigorous evaluation of the security of code-based cryptography. In practice, Prange’s, Lee and Brickell’s, and Dumer’s algorithms can be implemented relatively simply using bitwise operations such as exclusive or (XOR). These state-of-the-art algorithms [6], [7], [8], [9], [10], [11] are asymptotically faster; however, Dumer’s algorithm is faster for syndrome decoding problems with relatively small dimensions that can be solved. The overhead involved in improved algorithms cannot be ignored for McEliece cryptosystem with a relatively small (approximately 1,000) dimension that can actually be solved. Thus, we provide a multi-parallel GPU implementation of Dumer’s algorithm.

A graphics processing unit (GPU) is an integrated circuit that was originally developed to speed up image processing. In order to solve various problems in cryptography more efficiently, there have been attempts to study GPU-optimized algorithms. For the shortest vector problem (SVP), which is highly relevant to the lattice-based cryptography, GPU versions of the enumeration algorithm [12], [13], [14] and the sieving algorithm [15], [16] have been devised. For SDP, an FPGA implementation of the ISD algorithm has been proposed in [17]. However, to the best of our knowledge, there is no paper that specifically describes GPU-optimized ISD algorithms so far.

A. Contributions

In this paper, we propose the first GPU-optimized ISD algorithm, the multi-parallel Dumer’s algorithm. In the usual Dumer’s algorithm, a search algorithm such as depth-first search is used to enumerate columns of parity check matrix $H$. In contrast, our algorithm first expands all combinations of columns of $H$ onto the GPU memory. This allows all the combinatorial enumerations to be executed on the GPU memory in massively parallel, independently for each thread. For the number of parallels $N$, the computational cost of
our algorithm is $N$ times smaller than the usual Dumer’s algorithm. A unique feature of our algorithm is that the required GPU memory is constant for $N$.

In our experiments, we compared the runtime of the Dumer’s algorithm and our proposed algorithm on several parameter sets. As a result, our algorithm is tens of times faster than Dumer’s algorithm for all parameter sets. In addition, we attempted to break an unresolved SDP instance with parameter sets of $n = 1161$, $k = 929$, and $w = 22$ using our algorithm. By observing the behavior of our algorithm, it was confirmed that our algorithm is 251.2 times faster than Dumer’s algorithm for the SDP instance. The expected value of the runtime is 229.4 hours and we were able to finally solve it in 375.8 hours using NVIDIA Tesla V100 GPUs. Our new record has been posted on the decoding challenge in the Goppa-McEliece setting [18].

The rest of this paper is organized as follows. In Section II, we define the terms used in our paper. Section III gives a brief description of two representative ISD algorithms: Prange and Dumer’s algorithms. In Section IV, we propose multi-parallel Dumer’s algorithm, which is a GPU-oriented application of Dumer’s algorithm. We experiment and analyze the performance of our algorithm in Section V. Section VI concludes our paper.

II. PRELIMINARY

An $m$-dimensional vector is denoted by $h = (h_1, \ldots, h_m)$. The $i$-th element of $h$ is written as $h[i]$. We refer to the subsequence of the vector as $h[i : j] = (h_i, \ldots, h_j)$. The prefix (resp. suffix) of length $\ell$ of the vector is denoted by $h[1 : \ell]$ (resp. $h[-\ell : m]$) or simply $h[\ell]$ (resp. $h[-\ell]$). A matrix $H$ consisting of $n m$-dimensional column vectors is denoted by $H = (h_1, \ldots, h_n) \in \mathbb{F}_q^{n \times n}$. $H[i]$ represents the $i$-th $m$-dimensional vector $h_i$. $H^T$ is the transpose of a matrix $H$. $I$ denotes the identity matrix, and $O$ denotes the zero matrix. In what follows, we will only address the bit vector $h \in \mathbb{F}_2^m$ and the bit matrix $H \in \mathbb{F}_2^{n \times n}$. The Hamming weight for $x \in \mathbb{F}_2^m$ is denoted by $\text{wt}(x) = \{|i | x[i] = 1\}$. The SDP is defined as follows:

Definition 1 (SDP): For any integers $n, k$, and $w$ such that $k \leq n$ and $w \leq n$, consider a parity check matrix $H \in \mathbb{F}_2^{(n-k) \times n}$ and a syndrome $s \in \mathbb{F}_2^{n-k}$. Find an error vector $e \in \mathbb{F}_2^n$ of Hamming weight $\text{wt}(e) = w$ such that $eH^T = s$. The exclusive or (XOR) of two bit vectors $u \in \mathbb{F}_2^m$ and $v \in \mathbb{F}_2^m$ is written as $u \oplus v$.

III. SDP SET DECODING

Solving the SDP by computing an error vector $e$ can be viewed as choosing $w$ column matrices from $H$ and XORing them to ensure that the resulting bit string $eH^T$ is equal to $s$. The simplest algorithm for solving the SDP is the exhaustive search for such a combination of columns. The time complexity is obviously $\binom{n}{w}$. Below we introduce two principal ISD algorithms: Prange algorithm and Dumer’s algorithm.

A. Prange Algorithm

Prange algorithm [2] first performs column permutation and Gaussian elimination on the parity check matrix $H$. Namely, we compute $U_G(HU_P) = (I | Q)$ for two invertible matrices $U_P \in \mathbb{F}_2^{n \times n}$ and $U_G \in \mathbb{F}_2^{(n-k) \times (n-k)}$, where $U_P$ and $U_G$ denote the permutation matrix and the matrix corresponding to Gaussian elimination, respectively. Furthermore, $U_G$ is applied to syndrome $s$, we obtain $sU_G^T = \hat{s}$. If $\text{wt}(\hat{s}) = w$, then $e = \hat{e}U_P^{-1}$ is the solution for $\hat{e} = (\hat{s}, 0)$. Intuitively, this is the case when $w$ columns corresponding to the solution are all contained in the first $n-k$ columns of the permuted matrix $(I | Q)$, since if $\text{wt}(\hat{s}) = w$, then $\sum_{i \in \{x | s[x] = 1\}} I[i] = \hat{s}$. If $\text{wt}(\hat{s}) \neq w$, then start again from the beginning of the algorithm.

We consider the time complexity (also called the work factor: WF) of Prange algorithm. WF is approximated by $WF = K/P$, where $K$ is the computational cost required for a single iteration of the algorithm and $P$ is the probability of finding a solution in one iteration. $P = n(n-k)$ for Prange algorithm since the computational cost required for Gaussian elimination is dominant. The probability $P$ is given by $P = \binom{n-k}{w}/\min(2^{n-k}, \binom{n-k}{w})$. Therefore, $WF = n(n-k)/\min(2^{n-k}, \binom{n-k}{w})$.

B. Dumer’s Algorithm

Dumer’s algorithm [5] is a combination of the Lee and Brickell algorithm [3] and the birthday decoding algorithm. First, parity matrix $H$ and syndrome $s$ are preprocessed using column permutation and Gaussian elimination as in Prange algorithm. Note that Gaussian elimination is applied to matrix $H$ so that the rank of the matrix is $n-k-\ell$ for parameter $\ell > 0$ of the birthday decoding algorithm. Namely, $U_P$ and $U_G$ are applied to matrix $H$ as follows:

$$U_G(HU_P) = \begin{pmatrix} I_{n-k-\ell} & Q \\ O & \end{pmatrix},$$

where, $I_{n-k-\ell} \in \mathbb{F}_2^{(n-k-\ell) \times (n-k-\ell)}$, $O \in \mathbb{F}_2^{(n-k-\ell) \times \ell}$, and $Q \in \mathbb{F}_2^{\ell \times (n-k-\ell + \ell)}$. Similarly, we have $sU_G^T = \hat{s}$ by taking the product of the syndrome $s$ and $U_G$. Next, we consider dividing the random matrix $Q$ equally into left and right, $Q_1 \in \mathbb{F}_2^{\ell \times \frac{(n-k-\ell)}{2}}$ and $Q_2 \in \mathbb{F}_2^{\ell \times \frac{(n-k-\ell)}{2}}$. Then, $Q_1$ and $Q_2$ are further vertically divided into $n-k-\ell$ rows and $\ell$ rows. Namely, for the four matrices $Q_1' \in \mathbb{F}_2^{\ell \times \frac{(n-k-\ell)}{2}}$, $Q_2' \in \mathbb{F}_2^{\ell \times \frac{(n-k-\ell)}{2}}$, $Q_1'' \in \mathbb{F}_2^{\ell \times \frac{(k+1)}{2}}$, and $Q_2'' \in \mathbb{F}_2^{\ell \times \frac{(k+1)}{2}}$, namely,

$$Q = \begin{pmatrix} Q_1' & Q_2' \\ Q_1'' & Q_2'' \end{pmatrix}.$$  \hspace{1cm} (2)

The syndrome $\hat{s}$ is also split at the $n-k-\ell$ position in the row direction and is denoted as $\hat{s} = (s', s'')$. The error vector $e$ is split at the $n-k-\ell$ and $n-k-2\ell/2$ positions and denoted as $\hat{e} = (e_0, e_1, e_2)$. The above relationship is shown in Figure 1. Dumer’s algorithm enumerates the $p/2$ column combinations of $Q_1'$ and $Q_2'$ for parameter $p$ ($0 < p < w$). The combinations of columns are stored in $e_1$ and $e_2$ as bit
We assume that the XOR of the column combinations of \( Q_1' \) (resp. \( Q_2'' \)) as \( L_1 \) (resp. \( L_2 \)) as follows:

\[
L_1 = \{ e_1 \mid \text{wt}(e_1) = \frac{p}{2}\},
\]

\[
L_2 = \{ e_2 \mid \text{wt}(e_2) = \frac{p}{2}\}.
\]

We assume that the XOR of the column combinations \( e_1 \in L_1 \) and \( e_2 \in L_2 \) for \( Q_1' \) and \( Q_2'' \) is equal to \( s' \), namely,

\[
e_1Q_1'T \oplus e_2Q_2''T = s'.
\]

Such a pair \((e_1, e_2)\) can be computed by the birthday decoding algorithm, as described later. For \((e_1, e_2)\) satisfying Equation 5, we consider the XORs between the prefix of syndrome \( s'' \) and the column combination for \( Q_1' \) and \( Q_2'' \), which are the first \( n-k-\ell \) bits of matrix \( Q \):

\[
e_0 \leftarrow e_1Q_1'TT \oplus e_2Q_2''TT \oplus s''.
\]

If \( \text{wt}(e_0) = w-p \), then \( e = (e_0, e_1, e_2)U_P^{-1} \) is the solution.

We describe the pseudocode of Dumer’s algorithm and the birthday decoding in Algorithm 1 and 2. Algorithm 1 computes the set \( I \) of pairs \((e_1, e_2)\) that satisfy Equation 5. In Lines 2–4, we compute the set \( L_1 \) consisting of \( p/2 \) column combinations for the matrix \( Q_1' \). For each element \( e_1 \in L_1 \), the XOR of the last \( \ell \)-bit columns \( x \leftarrow e_1Q_1'T \) is computed. Then, we construct an associative array \( T \) by adding an element whose key is the bit string \( x \) and whose value is \( e_1 \). In Lines 5–8, we compute the set \( L_2 \) consisting of \( p/2 \) column combinations for the matrix \( Q_2'' \). For each element \( e_2 \) in \( L_2 \), the last \( \ell \)-bits XOR \( x \leftarrow e_2Q_2''TT \oplus s' \) are calculated. A pair \((e_1, e_2)\) satisfying Equation 5 is obtained by accessing the associative array \( T \) with \( x \) as the key.

Algorithm 2 represents the entire Dumer’s algorithm. In Lines 3–5, column permutation and Gaussian elimination on the parity matrix \( H \) and syndrome \( s \) are performed. Then, set \( I \) of pairs \((e_1, e_2)\) satisfying Equation 5 is obtained by the birthday decoding algorithm for \( Q_1' \) and \( Q_2'' \). Under Line 7, the candidate for the solution \((e_1, e_2)\) is verified using Equation 6. If solution \( e \) is found, the algorithm is terminated; otherwise, we restart from Line 2.

We describe the WF of Dumer’s algorithm. The computational cost required for one iteration of Dumer’s algorithm, \( K_{\text{dumer}} \), is given by using the most dominant cost in Dumer’s algorithm. The matching process in the birthday decoding algorithm (Lines 5–8 in Algorithm 1) is as follows:

\[
K_{\text{dumer}} = \left(\frac{k+\ell}{2}\right) \max \left\{ 1, \frac{(k+\ell)/2}{p/2} \right\}.
\]

The probability for one iteration \( P_{\text{dumer}} \) is given by the ratio of the total number of combinations of \( e_0, e_1 \) and \( e_2 \) to the combination to choose \( w \) columns from \( n \) columns as follows:

\[
P_{\text{dumer}} = \frac{(k+\ell)/2 \choose p/2 (n-k-\ell) \choose w-p}{C_n^w}.
\]
IV. MULTI-PARALLEL DUMER’S ALGORITHM

In this section, we propose a GPU-oriented application of Dumer’s algorithm. One of the most significant features of a GPU is its ability to massively parallelize and speed up simple processes such as computer graphics drawing or iterative processes that can be described with for loops. GPUs have many-core architectures with tens of thousands of cores capable of asynchronously executing processes in parallel using millions of threads. The simplest way to parallelize Dumer’s algorithm is instance-wise parallelization (parallelize in the while loop in Line 2 of Algorithm 2). However, the disadvantage of this approach is that it cannot withstand large-scale parallelization because the memory consumption increases with the number of threads. In addition, under the optimal parameter \( p \) in approximately 1000 dimensions in the McEliece cryptosystem, Dumer’s algorithm consumes more than 100 megabyte of memory for the associative array \( T \) used in the birthday decoding algorithm, even when the number of threads is 1.

To take advantage of the features of GPUs, it is appropriate to perform parallelization other than instance-wise parallelization. In this paper, we focused on parallelizing the birthday decoding algorithm form of Dumer’s algorithm. The for statements in Line 2 and Line 5 in Algorithm 1 are compatible with massively parallel processing on GPUs since they can be executed independently for each element. In preliminary experiments, it was found that the birthday decoding algorithm accounted for 99.3% of the running time required for the entire Dumer’s algorithm. Therefore, we chose to parallelize these processes of the birthday decoding algorithm using a GPU.

A. Architectures

In constructing our algorithm, we consider implementing it on CUDA (Compute Unified Device Architecture), which is a general-purpose parallel computing platform for GPUs developed by NVIDIA. For the sake of clarity, we will give some of the terms used in CUDA in the following. Basically, in GPU implementation, the data necessary for processing are passed from the CPU side to the GPU side, the main calculation is performed on the GPU, and the result is returned to the CPU. We refer to the CPU side as the host and the GPU side as the device. A function executed in parallel on a GPU is called a kernel. The architecture between the host and device is shown in Figure 2. We will GPU-optimize Dumer’s algorithm based on the following three design philosophies.

- Minimize device-host transfers: The hash array \( T \) and auxiliary arrays used to construct \( T \) are expanded on the device memory, and all processes related to the birthday decoding algorithm are executed on the device.
- Use as many threads as possible: Since the birthday decoding algorithm can be computed independently for each combination of columns, we use the number of combinations as the number of threads. The number of combinations (number of threads) is variable depending on the parameter \( \ell \) and \( p \).
- Adjust device memory usage to the appropriate size: A dynamic array (vector) is normally used for array \( T \) since the size of \( T[x] \) is indefinite (see Line 4 in Algorithm 1). However, since dynamic memory is not available in device memory, we consider constructing \( T \) as a fixed length array with the same space complexity as when using dynamic memory.

B. Entire Algorithm

The pseudocode of the multi-parallel Dumer’s algorithm is shown in Algorithm 3. The input to Algorithm 3 is one parity matrix \( H \), as in Algorithm 2, since the multi-parallel Dumer’s algorithm is not based on instance-wise parallelization. Algorithm 3 assigns a combination of \( p/2 \) columns of the matrix \( H \) to each thread of the GPU, and each thread performs the calculation on the Hamming distance to the solution. If an error vector \( e \) is found satisfying \( eH^T = s \)
and wt(e) = w, output e as the solution. We will explain each process of Algorithm 3 in detail. First, in Line 2, the MakeCombination function creates all $p/2$ combinations of indices for the matrices $Q_1$ and $Q_2$ and expands the patterns on the device memory as $\text{dL}$ and $\text{dR}$. Thus, we can refer to the $j$-th number ($1 \leq j \leq p/2$) of the $i$-th combination in lexicographic order as $\text{dL}[(i - 1) \cdot (p/2) + j]$. For instance, for the first left combination $\{76, 77, 78\}$ where $n = 431$, $k = 345$ and $\ell = 10$, we can refer to the second number as $\text{dL}[0 + 2] = 77$. Since the number of $p/2$ combinations of indices for the matrices $Q_1$ is $(k + \ell)/2$, the time complexity of the MakeCombination function is $O((p/2)((k + \ell)/2))$. The space complexity of the $\text{dL}$ is also $O((p/2)((k + \ell)/2))$. The same is true for $\text{dR}$. Although it may seem wasteful in terms of computational resources to expand all combinations in memory instead of iterating some variables, it is necessary so that each thread of the GPU can handle exclusive processing for each combination. Additionally, since the function is outside the while loop, it will be executed only once in the entire algorithm.

The details of the main procedure in the while loop are described below. In Line 7, the permuted matrices $\tilde{H}$ and $\tilde{s}$ are transferred to the device memory as $\text{dH}$ and $\text{ds}$ by the TransferToDevice function. In Line 9, the MultiParallelBirthdayDecoding function computes indices $i$ and $j$ corresponding to the combination of the column indices of matrices $Q_1$ and $Q_2$ that is the solution if $i$ and $j$ exist. Namely, $i$ and $j$ for $\text{dL}[(i - 1) \cdot (p/2) + 1 : i \cdot (p/2)]$ and $\text{dR}[(j - 1) \cdot (p/2) + 1 : j \cdot (p/2)]$. If such $i$ and $j$ are obtained, then in Line 11, the Reconstruct function computes $e_1$ and $e_2$ from $\text{dL}$ and $\text{dR}$. That is, for a set $\mathcal{A} = \{x \in \text{dL}[(i - 1) \cdot (p/2) + 1 : i \cdot (p/2)]\}$

$$e_1[x] = \begin{cases} 1 & (x \in \mathcal{A}) \\ 0 & (x \notin \mathcal{A}) \end{cases}.$$ 

$e_2$ is obtained in the same way. Then, $e_0$ and $e$ are computed in the same way as in Dumer’s algorithm; and if the solution $e$ is not found, the process starts over from the beginning of the while loop.

C. Multi-parallel Birthday Decoding Algorithm

This subsection describes the multi-parallel birthday decoding algorithm that performs the birthday decoding algorithm on the device memory. The pseudocode is given in Algorithm 4. In Line 2, we prepare some variables to be stored in the device memory. $S_1$ and $S_2$ of length $(k + \ell)/2$ store the bit string corresponding to the XOR of the column combination $\text{dL}$ and $\text{dR}$, namely, $S_1[i] = \sum_{x \in \mathcal{A}} Q_1[x]$ for $\mathcal{A} = \{x \in \text{dL}[(i - 1) \cdot (p/2) + 1 : i \cdot (p/2)]\}$ and $S_2[i] = \sum_{x \in \mathcal{A}} Q_2[x]$ for $\mathcal{A} = \{x \in \text{dR}[(j - 1) \cdot (p/2) + 1 : j \cdot (p/2)]\}$. $T$ of length $(k + \ell)/2$ is an alternative array for $T$ in the normal birthday decoding algorithm. $T[i]$ stores integers in the range $[1, (k + \ell)/2]$ and is used by $S_1[T[i]]$ as a pointer to $S_1$. $T$ is sorted by the $\ell$-bit suffix $S_1[T[i]][-\ell :]$ in lexicographical order. $T_{\text{start}}$ and $T_{\text{end}}$ of length $2^\ell$ are used to construct $T$ efficiently. $T_{\text{start}}[i]$ (resp. $T_{\text{end}}[i]$) stores the minimum (resp. maximum) index $x$ of $T$ such that $S_1[T[x]][-\ell :]$ is the $i$-th smallest $\ell$-bit string in lexicographical order. All values are set to 0 by the InitDeviceMemory function.

The first parallel for code block, Lines 3–7 corresponds to Lines 2–4 in Algorithm 1. Each combination $i$ is assumed to be executed asynchronously by each thread in the parallel for block. Therefore, the order of $i$ does not need to be considered. Note that there is a possibility that different threads refer to the same index of $\text{T}_{\text{start}}$ at the same time. In Line 7, we used CUDA atomic function (atomicAdd) to prevent the collisions, which allows other threads to avoid writing to the memory region being read. Each parallel for block is actually implemented as a kernel function. In Line 8, $S_1$ is constructed and $T_{\text{start}}[i]$ stores the number of indices $x$ of $S_1[x]$ such that $S_1[T[x]][-\ell :]$ is the $i$-th smallest 1-bit string. $T_{\text{start}}$ is copied to $T_{\text{end}}$. In Line 9, the ExclusiveScan function is applied to $T_{\text{start}}$. ExclusiveScan($x$) calculates the partial sums of $x$ for each range. For $y = \text{ExclusiveScan}(x)$, the partial sum of $x$ in range $[1, i)$, Therefore, $T_{\text{start}}$ is computed by the ExclusiveScan function. In Lines 10–12, $T$ and $T_{\text{end}}$ are calculated in parallel. Each block in $T$ corresponding to each $i$-th 1-bit string stacks the values to the left. Lines 13–19 correspond to the right enumeration and the left-right matching process in Lines 5–8 in Algorithm 1. In Line 17, we obtain the index $j$ from $T$ such that the corresponding 1-bit suffix in

---

**Algorithm 4: MultiParallelBirthdayDecoding**

**Input:** \( dL, dR, dH, ds, k, \ell, p \)

**Output:** \( I, J \)

1. \( I \leftarrow -1, J \leftarrow -1, n\text{Comb} \leftarrow \binom{k+\ell}{2}/p \)
2. \( S_1, S_2, T, T_{\text{start}}, T_{\text{end}} \leftarrow \) InitDeviceMemory\((S_1, S_2, T, T_{\text{start}}, T_{\text{end}})\)
3. **parallel for** \( i \leftarrow 1 \) to \( n\text{Comb} \)
   4. \( S_1[i] \leftarrow ds \)
   5. **for** \( j \in dL[(i - 1) \cdot (p/2) + 1 : i \cdot (p/2)] \) **do**
   6. \( S_1[i] \leftarrow S_1[i] + dH[j] \)
   7. \( T_{\text{start}}[S_1[i][-\ell :]] \leftarrow T_{\text{start}}[S_1[i][-\ell :]] + 1 \)
8. \( T_{\text{end}} \leftarrow T_{\text{start}} \)
9. \( T_{\text{start}} \leftarrow \text{ExclusiveScan}(T_{\text{start}}) \)
10. **parallel for** \( i \leftarrow 1 \) to \( n\text{Comb} \)
11. \( T[T_{\text{end}}[S_1[i][-\ell :]]] \leftarrow i \)
12. \( T_{\text{end}}[S_1[i][-\ell :]] \leftarrow T_{\text{end}}[S_1[i][-\ell :]] + 1 \)
13. **parallel for** \( i \leftarrow 1 \) to \( n\text{Comb} \)
14. \( S_2[i] \leftarrow 0 \)
15. **for** \( j \in dR[(i - 1) \cdot (p/2) + 1 : i \cdot (p/2)] \) **do**
16. \( S_2[i] \leftarrow S_2[i] + dH[j] \)
17. **for** \( j \in [T_{\text{start}}, S_2[i][-\ell :]], T_{\text{end}}[S_2[i][-\ell :]] \) **do**
18. **if** \( \text{wt}(S_1[T[j]]) \oplus S_2[i] = w - p \) **then**
19. \( I \leftarrow T[j], J \leftarrow i \)
20. **return** \( I, J \)
TABLE I: Parameter set for our experiments.

<table>
<thead>
<tr>
<th>Parameter set</th>
<th>n</th>
<th>k</th>
<th>w</th>
<th>ℓ</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>431</td>
<td>345</td>
<td>10</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>431</td>
<td>345</td>
<td>10</td>
<td>22</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>640</td>
<td>512</td>
<td>13</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>640</td>
<td>512</td>
<td>13</td>
<td>23</td>
<td>8</td>
</tr>
</tbody>
</table>

$S_i$ is $S_2[i][−ℓ⁺]$. If the Hamming distance between $S_1[T[j]]$ and $S_2[i]$ is $w − p$, then indices $T[j]$ and $i$ corresponding to the combinations are the solution.

We consider the time and space complexity of the multi-parallel birthday decoding algorithm. Let the number of parallel be $N$. Then, since the computational cost is dominated by left-right matching (Lines 13–19), the time complexity is $O(N^{-1}(k+\ell)/2)\max \{1, 2-\ell \} (k+\ell)/2)$. It is the cost of the multi-parallel Dumer’s algorithm $K_{n−\text{dumer}}$. The probability of the multi-parallel Dumer’s algorithm is $P_{\text{dumer}}$, WF of the multi-parallel Dumer’s algorithm is $K_{n−\text{dumer}} P_{n−\text{dumer}}^{-1}$. Ideally, it is minimized when $N = (k+\ell)/2$, but the maximum $N$ is at most several hundred in practice with the current performance of general-purpose GPUs. The space complexity is $O((p/2)\left(k+\ell/2\right)/2))$ dominated by the array $dL$ and $dR$ and is obviously independent of the parallel number $N$.

V. IMPLEMENTATION AND ANALYSIS

We ran the proposed algorithm on several SDP instances and measured the running time. We implemented all algorithms in CUDA 11.0 and C++14. All algorithms were run on an Intel Xeon E5-2686v4 server with an NVIDIA Tesla V100 (16 GB GPU memory). The maximum number of threads is 65535 × 65535 × 1024.

The parallel for blocks in Algorithm 4 were implemented using CUDA kernel functions. In order to simplify the implementation, all variables defined on the GPU were stored in CUDA global memory and no CUDA shared memory was used, which works faster for certain calculations [19], [20], [21].

First, we compared the running time of the normal Dumer’s algorithm (Algorithm 2) and the multi-parallel Dumer’s algorithm (Algorithm 3) for the parameter set shown in Table I. The results for each parameter set are shown in Table II. We gave up computing the solution for parameter 3 in Dumer’s algorithm. Time/#loops shows the running time consumed for 1 while loop in both algorithms. Since the overall running time is affected by luck, it is reasonable to compare the runtime of each algorithm using time/#loops. The results show that Algorithm 4 took much fewer time/#loops than Algorithm 2. For all parameters, our algorithm succeeded in increasing the speed by several tens of times, and the speed-up ratio for parameter 1 reached 82.8 times.

We analyze the solving time of Algorithm 3 theoretically. Let $N$ be the number of running threads and $P$ be the probability that a parallelized thread $\ell$ of the algorithm can find a solution in one iteration. Each probability variable $X_i (1 \leq i \leq N)$ denotes the number of iterations until the thread $(t_i)$ can find the solution; thus, $X = \min_i X_i$ denotes the overall execution time. The cumulative distribution function of $X$ is defined as $F(x) = \text{Pr}[X \leq x]$, which is given by

$$F(x) = 1 − (1 − P)^N,$$

The probability function of $X$ is given by $f(x) = \text{Pr}[X = x]$ and we have $f(x) = F(x) − F(x − 1) = (1 − P)^{N(x−1)}(1 − (1 − P)^N)$. Therefore, $X$ follows a geometric distribution with parameter $(1 − (1 − P)^N)$. The expected value of the execution time of Algorithm 3 is given by $E[X] = \sum_{x=1}^{\infty} x f(x)$, or

$$E[X] = \frac{1}{1 − (1 − P)^N},$$

and the variance of the end time of the decoding process $\text{Var}(X)$ is given by $\text{Var}(X) = E[X^2] − (E[X])^2 = \sum_{x=1}^{\infty} x^2 f(x) − (E[X])^2$, or

$$\text{Var}(X) = \frac{(1 − P)^N}{(1 − (1 − P)^N)^2}.$$

Next, we attempted to solve the decoding challenge for a parameter that has not yet been solved. The parameter set is $n = 1161$, $k = 929$ and $w = 22$. We searched the optimal parameter set for $p$ and $\ell$. For parameter $p$, we estimated the memory size required for Algorithm 3. The formula of our estimation is the multiplication of the space complexity $(p/2)\left(k+\ell/2\right)$ for Algorithm 3 and the size required for a single bit string (8 byte × block size $\lceil \frac{n−k}{16} \rceil$). The result is shown in Figure 3. We varied $n$ and $p$ and calculated the required memory for the optimal $\ell$ in each case. $n = 1161$ corresponds to the blue dashed line in the figure. In practice, the memory occupation for $n = 1161$ was 1MB when $p = 4$, and 500MB when $p = 6$. At higher $p$, our program did not run due to lack of memory. We confirmed that the parameter set of $p = 6$ and $\ell = 24$ are optimal both experimentally and theoretically. We set the total number of threads to $(k+\ell)/2) ≈ 17,861,900$. Then, we measured the time/#loops to be 0.051 seconds and the GPU memory consumption to be approximately 580 MB. Time/#loops is approximately 251.2
times faster than that of Dumer’s algorithm (without GPU or parallels) for the same parameter set. From a hardware perspective, Tesla V100 is capable of running 80 SMs (Streaming Multiprocessors) simultaneously, each consisting of 5120 cuda cores (threads). Therefore, our implementation performed about $251.2/80 = 3.14$ times faster per SM than the CPU implementation for $n = 1161$ instances. We derived that the expected value of the runtime for a 1161-dimensional SDP in the Goppa-McEliece setting is approximately 229.4 hours according to Eq. 9. We solved it in 375.8 hours in practice. Note that we have a one percent chance of solving the SDP within 1/10 of the expected time (2.29 hours) and a ten percent chance of solving within 1/100 of the expected time (2.29 hours).

VI. CONCLUSION

In this paper, we propose the multi-parallel Dumer’s algorithm, which is a GPU version of Dumer’s algorithm. Unlike naive parallel algorithms, our algorithm expands all the combinations of columns of the birthday decoding algorithm on the GPU’s memory, which allows us to perform massively parallel processing on the combinations executed independently for each thread. In addition, our algorithm can be applied to a variety of SDP instances since the number of parallelisms can be easily scaled without affecting the amount of data transfer between the GPU and CPU or the memory usage of the GPU. In experiments, our algorithm is approximately 250 times faster than the existing Dumer’s algorithm for a large parameter set $n = 1161$, $k = 929$ and $w = 22$. Our proposed algorithm solved a 1161-dimensional SDP in the Goppa-McEliece setting for the first time in approximately 380 hours.

REFERENCES