Graphics Processing Unit を用いた $GF(2^{32})$ 上の高速演算の実装

田中 哲士 †,‡　　安田 貴徳 ‡　　櫻井 幸一 †,‡

† 九州大学システム情報科学府,
819-0395 福岡市西区元岡 744 W2-712
‡ 九州先端科学技術研究所,
814-0001 福岡市早良区百地浜 2 丁目 1 番 22 号 福岡 SRP センタービル 7 階
tanasato@itslab.inf.kyushu-u.ac.jp
yasuda@isit.or.jp
sakurai@csce.kyushu-u.ac.jp

あらまし 有限体上の非線形な連立多元多項式の評価は多変数公開鍵暗号（MPKC）の暗号化，署名における重要なサブルーチンである．次数の大きな拡大体は，次数の小さな拡大体における同程度のセキュリティレベルの多元多項式の構成において，より高速であることが期待される．しかし，多元多項式の評価では多数の有限体上の加算及び乗算を行う必要がある．特に，拡大体上の乗算は複雑であり，高速な MPKC の構成の為には効率的な乗算を実現する必要がある．本論文では，中間体を利用した拡大体の構成により，$GF(2^{32})$ 上の乗算の効率化を図る．更に，Graphics Processing Unit を用いた実装を行い，その実装結果について CPU による実装結果と比較を行う．

Implementation of Efficient Operations over $GF(2^{32})$ using Graphics Processing Unit

Satoshi Tanaka†,‡　　Takanori Yasuda‡　　Kouichi Sakurai†,‡

†Kyushu University,
W2-712, Motooka 744, Nishi-ku, Fukuoka, 819-0395, JAPAN
‡Institute of Systems, Information Technologies and Nanotechnologies,
tanasato@itslab.inf.kyushu-u.ac.jp
yasuda@isit.or.jp
sakurai@csce.kyushu-u.ac.jp

Abstract Evaluating non-linear multivariate polynomial systems over finite fields is an important subroutine, e.g., for encryption and signature verification in multivariate cryptography. The security of multivariate cryptography definitely becomes lower if a larger field is used instead of $GF(2)$ given the same number of bits in the key. However, we still would like to use larger fields because multivariate cryptography tends to run faster at the same level of security if a larger field is used. In this paper, we compare the efficiency of several techniques for evaluating multivariate polynomial systems over $GF(2^{32})$ vi their implementations on graphics processing units.

1　はじめに

多変数多項式暗号は，有限体上の連立多次多変数多項式の評価を一種の写像として用い，暗号化処理の一部に利用する方式である．元々，多変数多項式暗号は公開鍵暗号の一種 [3] であったが，現在では高速な署名方式 [2] として期待されているだけでなく，公開鍵暗号レベルの証明可能安全性を有する秘密鍵暗号 [1] も提案されている．多変数多項式暗号は有限体上の連立多次多元方程式の解決問題 (MP 問題) を安全性の根拠としている．MP 問題は NP 完全であることが知られているため，多変数多項式暗号は量子コンピュータによる解読に耐性のある暗号としても期待されている．

有限体上の連立多次多変数多項式の評価では，多数の有限体上の加算，乗算の計算が必要となる．従って，これらの計算の効率化が多変数多項式暗号全体の効率化に繋がる．一方で，多変数多項式暗号の安全性は連立多次多変数多項式の変数，多項式の数，使用する有限体の位数の大きさに依存する．故に，使用する有限体の位数の大きさが大きければ，変数，多項式の数が同規模の連立多次多変数多項式よりも安全性が高くなるが，連立多次多変数多項式の評価の計算コストが大きくなる．

多変数多項式暗号でよく用いる有限体の一つとして $GF(2)$ もしくはその拡大体が挙げられる．これは $GF(2)$ 及びその拡大体は，加算を排他的論理和のみで実装できるため，剰余演算が必要なく，高速に計算を行うことができる為である．現在は $GF(2), GF(2^2), GF(2^4), GF(2^8)$ 等が利用されているが，将来的に，より巨大な位数を持つ有限体を利用する可能性がある．しかし，巨大な拡大体上における乗算は計算コストが高く，効率的な乗算手法を見つけることが必要となってくる．

本研究では，標数 2 の拡大体である，$GF(2^{32})$ の演算，特に計算コストのかかる乗算演算の効率的な手法について比較検討を行う．また，比較検討した手法について CPU 及び GPU (Graphics Processing Unit) を用いて実装し，検証を行う．

2　拡大体の演算

本章では，有限体の拡大と拡大体の基本となる演算，即ち加算と乗算の手法について説明を行う．

2.1　有限体の拡大

素数 p に対し，体 $F = GF(p^m), K = GF(p^n)$ $(m, n \geq 1)$ とする．このとき，$m \mid n$ であれば K は F の拡大体である．また，F, K は共に $GF(p)$ の拡大体であり，F は $GF(p)$ と K の中間体となる．

以下，$m \mid n$ とし，$k = n/m$，$q = p^m$ とおく．このとき，体の拡大 K/F に対して，Frobenius 写像 σ_0 が以下のように定義される．

$$\sigma_0(a) = a^q.$$

これは K から K への写像であり，以下の性質を満たす．

1. $\sigma_0(a + b) = \sigma_0(a) + \sigma_0(b)$ $(a, b \in K)$,
2. $\sigma_0(ab) = \sigma_0(a)\sigma_0(b)$,
3. $\sigma_0(\alpha) = \alpha$ $(\forall \alpha \in F)$.

即ち，σ_0 は K の加算及び乗算を保つ写像であり，F の元を不変にする．更に，σ_0 は全単射写像であることが知られており，環として同型写像である．

このとき，Galois 群 $\mathrm{Gal}(K/F)$ が次のように与えられる．

$$\mathrm{Gal}(K/F) = \{\sigma : K \mapsto K | \sigma(\alpha) = \alpha (\forall \alpha \in F)\}.$$

これは写像の合成により群をなす．従って，Frobenius 写像はガロア群の元となる．更に，このガロア群は Frobenius 写像が生成する巡回群である為，

$$\mathrm{Gal}(K/F) = \{\sigma_0^0, \sigma_0, \sigma_0^2, \ldots, \sigma_0^{k-1}\}.$$

となる．この群位数は k である．

2.2 拡大体の加算

K は F 上の k 次未満の多項式の集合として表現できる．ここで，F 上 k 次既約多項式を，

$$f_0(x) = x^k + a_{k-1}x^{k-1} + \cdots + a_1x + a_0 \quad (a_0, \ldots, a_{k-1} \in F), \quad (1)$$

とすれば，K は次のように表現可能である．

$$K = \{c_{k-1}x^{k-1} + \cdots + c_1x + c_0 | c_0, \ldots, c_{k-1} \in F\}.$$

e_1, e_2 を K の元とすれば，加算 $e_1 + e_2$ は次のように記述される．

$$e_1 + e_2 := e_1(x) + e_2(x) \mod f_0(x).$$

従って，加算は単純に多項式の係数の和によって計算できる．

2.3 拡大体の乗算

拡大体の乗算は加算とは異なり一般には単純に計算できない．以下に，拡大体の乗算手法について説明する．

2.3.1 多項式法

拡大体の加算と同様に，K を F 上の多項式の集合と考えたとき，K 上の元 e_1, e_2 の乗算は次のように記述される．

$$e_1 * e_2 := e_1(x) * e_2(x) \mod f_0(x) \quad (2)$$

2.3.2 巡回群法

一般に，K に対して $K^* := K \setminus \{0\}$ とすれば，K^* は乗算に関して巡回群となることが知られている．従って，ある $\gamma \in K^*$ が存在し，$K = \langle\gamma\rangle$ となる．この γ を固定すると，K^* 上の任意の元は γ^l (l は整数) と表現できる．特に，$0 \le l \le p^n - 2$ の範囲では l は一意的に定まる．従って，K^* は $[0, p^n - 2]$ と同一視することができる．この時，K^* 上の乗算は，$p^n - 1$ を法とした和と一致する．

2.3.3 正規基底法

一般に，有限次ガロア拡大 K/F に対して，ある $\alpha \in K$ が存在し，$\{\sigma(\alpha) | \sigma \in \mathrm{Gal}(K/F)\}$ は K の F-基底となる．これを K/F の正規基底と呼ぶ．

K 対して，α を上記のようにとると，正規基底は次のように表現できる．

$$\{\alpha, \alpha^q, \alpha^{q^2}, \ldots, \alpha^{q^{k-1}}\}. \quad (3)$$

このとき，K 上の任意の元 a は次のように一意に表すことが可能である．

$$a = c_0\alpha + c_1\alpha^{p^m} + \cdots + c_{k-1}\alpha^{p^{(k-1)m}} \quad (c_0, \ldots, c_{k-1} \in F). \quad (4)$$

K 上の元 a を式 4 と同様とし，これを $a = [c_0, c_1, \ldots, c_{k-1}]_n$ と表現こととする．このとき，Frobenius 写像 $\sigma_0(a)$ は次のように計算できる．

$$\sigma_0(a) = a^q = [c_{k-1}, c_0, c_1, \ldots, c_{k-2}]_n. \quad (5)$$

従って，この計算は右巡回シフトとして表現される [4]．$a = [c_0, c_1, \ldots, c_{k-1}]_n$, $b = [c'_0, c'_1, \ldots, c'_{k-1}]_n$ を K 上の元として，乗算 ab の結果を $[d_0, d_1, \ldots, d_{k-1}]_n$ とする．このとき，各 d_i $(i = 0, \ldots, k-1)$ は $c_0, c_1, \ldots, c_{k-1}, c'_0, c'_1, \ldots, c'_{k-1}$ の F 上の 2 次多項式で表現可能である．この 2 次多項式を $p_i(c_0, c_1, \ldots, c_{k-1}, c'_0, c'_1, \ldots, c'_{k-1})$ とする．ここで，$\sigma_0(ab)$ を計算すると，式より次のように表現できる．

$$\sigma_0(ab) = [d_{k-1}, d_0, \ldots, d_{k-2}]_n. \quad (6)$$

一方，Frobenius 写像の性質から $\sigma_0(ab) = \sigma_0(a)\sigma 0(b)$ でもあるため，

$$\begin{aligned} & \sigma_0(ab) \\ = \ & \sigma_0(a)\sigma_0(b) \\ = \ & [c_{k-1}, c_0, \ldots, c_{k-2}]_n * [c'_{k-1}, c'_0, \ldots, c'_{k-2}]_n \\ = \ & [p_0(c_{k-1}, c_0, \ldots, c_{k-2}, c'_{k-1}, c'_0, \ldots, c'_{k-2}), \ldots, \\ & \ p_{k-1}(c_{k-1}, c_0, \ldots, c_{k-2}, c'_{k-1}, c'_0, \ldots, c'_{k-2})]_n. \end{aligned}$$

となる．従って，式 6 と比較すれば d_{k-2} は次のように表現できる．

$$d_{k-2} = p_{k-1}(c_{k-1}, c_0, \ldots, c_{k-2}, c'_{k-1}, c'_0, \ldots, c'_{k-2}).$$

表 1: 各乗算法の評価.

乗算法	加算コスト	乗算コスト	メモリ
多項式法	簡単	難しい	少ない
巡回群法	難しい	簡単	多い
正規基底法	簡単	難しい	少ない
乗算表利用	簡単	簡単	多い

同様に，$\sigma_0^2(ab), \sigma_0^3(ab), \ldots$ と考えると，d_i $(\forall i)$ は 2 次多項式 p_{k-1} 及び右シフト演算のみで表現可能である．従って，ab の正規基底の係数 $d_0, \ldots, d_{k-1}$ は 1 つの 2 次多項式 p_{k-1} と右巡回シフトを利用して計算することができる．

2.4 乗算表を利用した手法

この手法は，他の手法と異なり実装時に用いる手法である．K 上の乗算を他の手法を用いて事前計算し，あらかじめ全ての元の組み合わせに対応する乗算表を求める方式である．乗算はする際は単純に二つの元が示す乗算表の位置を参照すればよい．

2.5 各乗算法の評価

これまでに述べた，各乗算法について評価を行う．K 上の計算を行う上で，加算，乗算の計算コスト及び計算で使用するメモリ量について評価したものを表 1 に示す．

2.5.1 多項式法

K 上の元 e_1, e_2 をそれぞれ，$e_1 = c_{k-1}x^{k-1} + \cdots + c_1x + c_0$，$e_1 = {c'}_{k-1}x^{k-1} + \cdots + {c'}_1x + {c'}_0$ $(c_0, \ldots, c_{k-1}, {c'}_0, \ldots, {c'}_{k-1})$ とし，既約多項式 f_0 を式 1 のように定義する．このとき，二つの元の加算は各次数の係数について加算を行えばよい為，k 回の F 上加算で計算できる．一方，二つの元の乗算は $e_1 * e_2 = c_{k-1}{c'}_{k-1}x^{2k-2} + \cdots + c_0{c'}_0 \mod f_0(x)$ である．従って，$(k-1)^2$ 回の F 上加算及び k^2 回の F 上乗算，1 回の K 上の剰余演算が必要となる．また，1 つ既約多項式が必要となるため，$k\lceil \log_2 p^m \rceil \simeq n\lceil \log_2 p \rceil$ ビットのメモリ量が必要となる．

2.5.2 巡回群法

巡回群法では，K 上の乗算は 1 回の加算及び $k-1$ を法とした剰余演算が必要となる．しかし，巡回群法を利用する場合，二つの元の加算が困難である．そこで，実際には加算では多項式法を利用し，乗算では多項式を巡回群上の元に置き換えることで実現する．多項式と巡回群上の元の置き換えは多項式から巡回群への写像と巡回群から多項式への写像を表す 2 つの置換表を利用する事で実現できる．従って，加算は多項式法と同様に k 回の F 上加算で計算可能であり，乗算は 1 回の加算及び $k-1$ を法とした剰余演算，そして 2 回の置換表参照が必要となる．また，置換表は K 上の元に対する写像を用意する必要があるため，$2p^n\lceil \log_2 p^n \rceil$ ビットのメモリ量が必要となる．

2.5.3 正規基底法

正規基底法における K 上加算は多項式法と同様に k 回の F 上加算で実現できる．一方で，K 上二つの元 $a = [c_0, \ldots, c_{k-1}]_n$，$b = [{c'}_0, \ldots, {c'}_{k-1}]_n$ の乗算 $a*b$ は，$2(k-1)$ 回の右巡回シフト演算と 2 次多項式 $p(c_0, \ldots, c_{k-1}, {c'}_0, \ldots, {c'}_{k-1})$ の評価を k 回行う事で計算できる．この多項式は k^2-1 回の F 上加算と $2k^2$ 回の F 上乗算が必要となる．しかし，F 上乗算の内，$c_i{c'}_j$ $(0 \le i, j \le k-1)$ は共通である為，事前に計算する事で計算を省略できる．また，最適な正規基底を選択する事で，$i < j$ となる $c_i, c_j, {c'}_i, {c'}_j$ に対して，

$$\begin{aligned} & p(c_0, \ldots, c_{k-1}, {c'}_0, \ldots, {c'}_{k-1}) \\ = \; & c_0{c'}_0 + \textstyle\sum_{0 \le i < j < k} s_{i,j}(c_i + c_j)({c'}_i + {c'}_j) \\ & (\forall (i,j), s_{i,j} \in F), \end{aligned}$$

となる．従って，正基底法における K 上の乗算は $k(k-1)(k+2)/2$ 回の F 上加算，$k(k^2+1)/2$ 回の F 上乗算，$2(k-1)$ 回の右巡回シフト演算で計算できる．また，使用するメモリとして $(k^2-k+2)\lceil \log_2 p^m \rceil/2$ ビット必要となる．

2.5.4 乗算表を利用した手法

多項式法で事前に乗算表を構成していると仮定する．このとき，K 上加算は多項式法と同様に k 回の F 上加算で計算できる．一方で，K 上の乗算は 1 回の乗算表の参照で計算が可能である．しかし，乗算表では全ての乗算結果を用意する必要があるため，$p^{2n}\lceil \log_2 p^n \rceil$ ビットのメモリ空間が必要となる．

3 GPGPU

GPU (Graphics Processing Unit) は本来，コンピュータの画像処理を目的として使用される計算機である．しかしながら，近年のオンラインネットワークゲーム等の高解像度な CG を利用するソフトウェアが要求する画像処理能力の水準が上がるにつれて，GPU の計算能力は急速に発達した．これにより，現在では GPU は単純な計算において，CPU よりも強力な計算能力を持つようになった．

これらの発達した GPU の計算能力を画像処理以外の一般的な問題に対して利用する事を目的とした技術が GPGPU (General-Purpose computing on GPUs) である．GPU は SIMD に基づいた設計をされており，単純な構成の複数のプロセスを同時に処理する事に適している．一方で，GPU が有する個々の計算コアの性能は CPU のコアよりも低く，連続的な処理には適していない．その為，GPU 上で実装を行う際は可能な限り並列化したアルゴリズムの実現が重要となる．

3.1 CUDA API

CUDA は NVIDIA が提供している C 言語をベースとした NVIDIA 製の GPU 向けの開発言語である [5]．CUDA が提供される以前から，GPU でプログラミングを行うようなツールや API は存在した．しかしながら OpenGL や Direct X といったそれらの API は画像処理を目的としていた為，GPGPU に利用する為には特殊な技術を利用する必要があり，効率が悪かった．CUDA は GPU の計算コアを画像描写を介さずに使用することができる為，効率的な実装が可能である．

CUDA ではグラフィックスカード等のデバイスをコンピュータ等のホストが制御して動作する．ホストがデバイス側に実行指示する関数等の機能をカーネルと呼ぶ．カーネルは複数のスレッドを並列に実行し，処理を行う．しかしながら，スレッドが個々に動作するとメモリ等のデータ共有が困難となるため，ブロックと呼ばれるスレッドよりも大きな構成単位を利用している．ブロック内では，メモリ等のデータ共有が行われる．また，同時に処理するスレッドの単位としてワープが存在する．スレッドはワープ単位で並列に実行される．実際にはワープは自動的に構成されるため，実装上で意識することは少ない．

4 $GF(2^{32})$ 上の乗算

$GF(2^{32})$ 上で乗算に必要な計算コストとメモリ量を表 2 に示す．多項式法と正規基底法は必要なメモリ量は少ないが，非常に多くの計算回数が必要であり計算時間がかかる．一方で，乗算表を利用した手法は計算に参照 1 回で計算できるものの，必要なメモリ量が 64EB となっており，現実的に実現不可能となっている．巡回群法も同様に計算回数は効率的であるものの，メモリ量が 32GB 必要であり現実的な構成ではない．

4.1 中間体を利用した体の拡大

乗算表やを用いた乗算法は計算回数が少なく，効率的であるが，$GF(2^{32})$ 上ではメモリサイズの問題により実装困難である．しかし，$GF(2^8)$ 上の乗算表に必要なメモリサイズは 64kB であり実現可能なサイズとなる．また，$GF(2^{16})$ 上の乗算表は 8GB のメモリ量が必要だが，巡回群法に必要なメモリサイズは 256kB である．そこで，有限次ガロア拡大 $GF(2^{32})/GF(2)$ に対して，中間体を利用することによる乗算手法が考えられる．例として，$GF(2^8)$ を利用する場合を考える．この手法では，$GF(2^8)/GF(2)$

表 2: $GF(2^{32})$ 上の乗算のコスト.

$GF(2^{32})$ 上の乗算法	計算コスト	メモリ
多項式法	XOR961 回，AND1024 回，剰余 1 回	4B
巡回群法	加算 1 回，剰余 1 回，参照 3 回	32GB
正規基底法	XOR16864 回，AND16400 回，右巡回シフト 62 回	63B
乗算表利用	参照 1 回	64EB

に対して $GF(2^8)$ 上の乗算を乗算表で実現し，$GF(2^{32})/GF(2^8)$ に対する $GF(2^{32})$ 上の乗算を多項式法又は正規基底法を用いて実現する．このとき，$GF(2^{32})/GF(2^8)$ に対して $k = 32/8 = 4$ である為，多項式法では，$GF(2^{32})$ 上の乗算を 9 回の $GF(2^8)$ 上の加算 (72 回の排他的論理和)，16 回の $GF(2^8)$ 上の乗算表参照，1 回の剰余演算で計算可能であり，正規基底法では，$GF(2^{32})$ 上の乗算を 288 回の排他的論理和，34 回の $GF(2^8)$ 上の乗算表参照で実現できる．

同様に，$GF(2^2)$，$GF(2^4)$ を中間体として利用することも考えられる．中間体を利用した場合の $GF(2^{32})$ の乗算コストを表 3 に示す.

4.2 $GF(2^{32})$ 上の計算コスト

$GF(2^{32})$ 上で乗算に必要な計算コストとメモリ量を表 2 に示す.

5 実装実験

5.1 実装環境

$GF(2^{32})$ 上の効率的な乗算手法を確認する為に，実装実験を行う．本実験では OS として Ubuntu 10.04 LTS 64bit, CPU に Intel Core i7 875K を，GPU に Nvidia Ge Force 580 GTX を利用した．また，メモリは DDR3 8GB となっている.

5.2 実験内容

$GF(2^{32})$ 上の乗算法について，以下の手法について CPU 及び GPU 上で実装を行い，ランダムな $GF(2^{32})$ 上の元を用いた乗算をそれぞれの手法で 67,108,864 回行い，その時間を測定する.

1. 乗算表+多項式法:
   $GF(2^{32})/GF(2^k)/GF(2)$ $(k = 1, 2, 4, 8)$
2. 乗算表+正規基底法:
   $GF(2^{32})/GF(2^k)/GF(2)$ $(k = 1, 2, 4, 8)$
3. 巡回群法+多項式法:
   $GF(2^{32})/GF(2^{16})/GF(2)$
4. 巡回群法+正規基底法:
   $GF(2^{32})/GF(2^{16})/GF(2)$

また，多項式法における各体の原始多項式として以下のものを利用した.

1. $GF(2^{32})/GF(2)$:
   $Y^32 + Y^22 + Y^2 + Y + 1 = 0$
2. $GF(2^{32})/GF(2^2)/GF(2)$:
   $Y^16 + Y^3 + Y + X = 0$
3. $GF(2^{32})/GF(2^4)/GF(2)$:
   $Y^8 + Y^3 + Y + X = 0$
4. $GF(2^{32})/GF(2^8)/GF(2)$:
   $X^4 + Y^2 + (X + 1)Y + (X^3 + 1) = 0$
5. $GF(2^{32})/GF(2^{16})/GF(2)$:
   $Y^2 + Y + X^{13} = 0$

表 3: 中間体を用いた $GF(2^{32})$ 上の乗算のコスト.

中間体	乗算法		計算コスト				メモリ
	中間体/$GF(2)$	$GF(2^{32})$/中間体	XOR	参照	剰余	加算	
$GF(2^2)$	乗算表	多項式法	450	1024	1	-	6B
		正規基底法	4320	16400	-	-	62B
$GF(2^4)$		多項式法	196	64	1	-	130B
		正規基底法	144	1120	-	-	156B
$GF(2^8)$		多項式法	72	16	1	-	64kB+4B
		正規基底法	288	34	-	-	64kB+4B
$GF(2^{16})$	巡回群法	多項式法	1	12	4	4 + 1	256kB+4B
		正規基底法	8	15	5	5	256kB+4B

表 4: 多項式法を用いた乗算法の CPU における実装結果.

拡大形式	中間体の乗算	計算時間
$GF(2^{32})/GF(2)$	乗算表	339.667s
$GF(2^{32})/GF(2^2)/GF(2)$		121.596s
$GF(2^{32})/GF(2^4)/GF(2)$		32.380s
$GF(2^{32})/GF(2^8)/GF(2)$		8.681s
$GF(2^{32})/GF(2^{16})/GF(2)$	巡回群法	3.532s

5.3 実験結果

多項式法を用いた乗算の CPU 実装結果を表 4 に示す．GPU 実装結果及び，正規基底法を用いた結果については当日発表を行う．

6 まとめ

本論文では $GF(2^{32})$ 上で最も高速となる乗算の手法を比較検討した．実装実験では CPU 実装において，中間体として $GF(2^{16})$ の巡回群法による乗算を利用したものが最も高速であった．しかし，本論文では，あくまで $GF(2^{32})$ 上における高速な計算手法しか評価を行っておらず，有限体上の高速な計算手法に対して一般的な見解をものているものではない．従って，$GF(2)$ の一般の拡大体，更には一般の有限体について効率的な計算手法の評価を与えることが今後の課題である．

謝辞

本研究は一部において，日本学術振興会科学研究費助成事業若手研究 B（24740078）及び，総務省戦略的情報通信研究開発推進事業（SCOPE）ICT イノベーション創出型研究開発フェーズ□『多変数多項式システムを用いた安全な暗号技術の研究』の助成を受けている．

参考文献

[1] Berbain, B., Gilbert, H., Pataring, J.: QUAD: a Pratical Stream Cipher with Provable Security. In EUROCRYPTO'06, LNCS, vol.4004, pp.109-128, Springer, 2006.

[2] Jintai, D., Dieter, S.: Rainbow, a new multivariable polynomial signature scheme. In Applied Cryptography and Network Security - ACNS 2005, LNCS, vol. 3531, pp.164-175, Springer, 2005.

[3] Matsumoto, T., Imai, H.: Public quadratic polynomial-tuples for efficient signature-varification and message-encryption. LNCS, vol.330, Proceedings of EUROCRYPT'88, pp. 419-453, Springer, 1988.

[4] Menezes, A.J., van Oorschot, Vanstone, S.A.: Handboook of Applied Cryptography, CRC Press 1997.

[5] NVidia Developer Zone
`https://developer.nvidia.com/category/zone/cuda-zone`