多変数暗号における GPU を用いた高速実装手法の評価

田中 哲士† 西出 隆志† 櫻井 幸一†

†九州大学システム情報科学府，九州先端科学技術研究所
819-0395 福岡市西区元岡 744 W2-712
tanasato@itslab.csce.kyushu-u.ac.jp
nishide@inf.kyushu-u.ac.jp
sakurai@csce.kyushu-u.ac.jp

あらまし 多変数暗号は安全性の根拠に NP 困難である有限体上の多変数の方程式の解決困難性を利用しており，ポスト量子コンピュータ暗号として期待されている．しかしながら，多変数暗号は計算コストが大きく，この計算コストを削減した高速な実装が要求されている．本発表では GPU を利用した多変数暗号の効率的な実装手法について提案し，その効果について評価を行う．GPU は高機能なグラフィクス処理を行う為に高い演算能力を有しており，汎用的並列演算装置としての活用が期待されている．提案手法では多変数暗号のアルゴリズムを並列化し，GPU の処理に最適化させる事で更なる効率化を図っている．

Evaluation of Efficient Implementation of Multivariate Cryptography with GPU

Satoshi Tanaka† Takashi Nishide† Kouichi Sakurai†

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

Abstract Multivariate cryptography is based on multivariate polynomials over finite fields. It is considered to be a good candidate of post-quantum cryptography, because the problem of finding a solution to a polynomial system over finite fields is NP-Hard. However, multivariate cryptography has a heavy computational cost and an efficient implementation, which has a reduced computational cost, is required. In this paper, we propose an efficient implementation of multivariate cryptography with GPU and evaluate efficiency of the proposal. GPU is considered to be a commodity parallel arithmetic unit. Our proposal parallelizes an algorithm of multivariate cryptography, and it realizes efficiency by optimizing the algorithm with GPU.

1 イントロダクション

QUAD ストリーム暗号は多変数多項式を疑似乱数の生成に利用しており，安全性の根拠に有限体上における二次の多変数多項式の求解問題（MQ 問題）の解決困難性を利用している．MQ 問題は一般に NP 困難であることが知られている．その為，QUAD は公開鍵暗号に準じた

証明可能安全性を有する．しかしながら，一方で QUAD はキーストリームの生成に多変数多項式の計算が必要であるため，他の共通鍵暗号に比べ暗号化の速度が遅いという欠点を持つ．

本論文では，QUAD ストリーム暗号の実用性を上げるため高速実装を行い，その高速性について評価する．実装では GPGPU と呼ばれる GPU を一般的な用途に利用する手法を用いる．

2 QUAD ストリーム暗号

QUAD ストリーム暗号は 2006 年に Berbain らによって提案されたストリーム暗号である [2]．二次の多変数多項式を疑似乱数生成器として利用し，キーストリームを生成する．本章では，QUAD が利用する有限体上における二次の多変数多項式と QUAD の構成，アルゴリズムについて解説する．

2.1 有限体上における多変数多項式

複数の変数で構成された多項式を多変数多項式と呼ぶ．QUAD ストリーム暗号は式 (1) で表現されるような二次の多変数多項式を利用する．このとき，$x_i, x_j (1 \leq i \leq j \leq n)$ は変数 $\alpha_{i,j}$, β_i, γ は多項式の係数を意味する．

$$\sum_{1 \leq i \leq j \leq n} \alpha_{i,j} x_i x_j + \sum_{1 \leq i \leq n} \beta_i x_i + \gamma \quad \cdots\cdots\cdots\cdots\cdots\cdots ①$$

QUAD は多変数二次多項式を関数として利用する．即ち，式 (1) に表わされるような多変数二次多項式を関数 $f(x_1, \ldots, x_n)$ として取り扱う．更に，複数の関数を一つのシステム MQ としてまとめ，式 (2) のように取り扱う．ここで，$\boldsymbol{x} = (x_1, \ldots, x_n)$ とする．

$$MQ(\boldsymbol{x}) = \{f_1(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x})\} \quad \cdots\cdots\cdots\cdots\cdots\cdots ②$$

QUAD ストリーム暗号は有限体上の多変数多項式の求解問題を安全性の根拠としている．すなわち，MQ から得られた関数値から，元の変数 $\boldsymbol{x} = (x_1, \ldots, x_n)$ を求める問題である．有限体上の多変数多項式の求解問題において二次のものを MQ 問題と呼ぶ．MQ 問題は任意の $\mathbb{GF}(q)$ に対して NP 困難であることが知られている [5]．

2.2 QUAD の構成

QUAD ストリーム暗号は MQ を疑似乱数生成器として利用している．一般に，$\mathbb{GF}(q)$ 上で n 個の変数 $\boldsymbol{x} = (x_1, \ldots, x_n)$ 及び，m 個の関数で構成されるシステム $(\boldsymbol{x}) = \{f_1(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x})\}$ で構成される QUAD を QUAD(q,n,r) と表現する．このとき，r は出力されるキーストリームの数であり，通常 n の倍数で表現される．すなわち，$r = m - n$ であり，$r = (k-1)n$ である．k については，多くの場合 $k = 2$ であるため，$r = n$ となる．

QUAD のキーストリーム生成は以下の三つのステップで構成される．

計算ステップ
　システム $S(\boldsymbol{x})$ の値を計算する．

出力ステップ
　求めた値の内，$f_{n+1}(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x})$ の値をキーストリームとして出力する．

更新ステップ
　求めた値の内，$f_1(\boldsymbol{x}), \ldots, f_n(\boldsymbol{x})$ の値を次の $\boldsymbol{x}$ として更新する．

QUAD ストリーム暗号におけるキーストリーム生成のイメージを図 1 に示す．更新ステップで値が変更される変数 $\boldsymbol{x} = (x_1, \ldots, x_n)$ を基に，次々とキーストリームを出力するイメージとなる．

2.3 QUAD の計算回数

QUAD(q,n,r) における 1 回のキーストリーム生成に必要な計算量は以下の通りになる．多変数二次多項式 $f_k(\boldsymbol{x})$ の計算には二次の項の乗算とすべての項の加算が必要となる．n 個の変数における二次の項の数は $n(n-1)/2$ で表現される．従って，多変数二次多項式の計算は $n(n-1)/2$

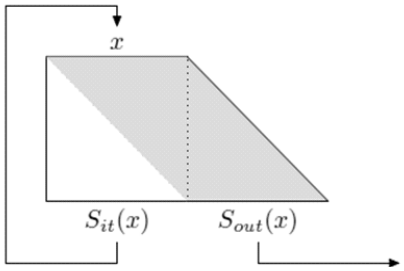

図 1: キーストリームの生成 [2]

回の乗算が，加算では更に一次の項，定数項が存在するため，$n(n+1)/2-1$ 回の加算が必要となる．QUAD(q,n,r) では $m=n+r$ 個の多変数二次多項式を利用する．このとき，$m=kn$ であるため，例えば，QUAD$(2,320,320)$ においては一回の多項式の計算に 51040 回の乗算と 51359 回の加算が必要となる．実際には 640 個の多項式があり計算回数はその 640 倍となる．

2.4 QUAD のセキュリティ

QUAD のセキュリティは MQ 問題の解決困難性に基づいている．具体的なセキュリティとして 2007 年に Yang らが行った XL-Widedemann アルゴリズムに基づく解析により，Bertain らが提案した QUAD(256,20,20)， QUAD(16,40,40), QUAD(2,160,160) は計算量的に安全でないことが示された [6]．実際には QUAD(2,160,160) よりも大規模な構成である QUAD(2,256,256) や，QUAD(2,320,320) を利用する必要がある．

3 CUDA コンピューティング

3.1 GPGPU

Graphical Processing Unit(GPU) は本来，コンピュータの画面出力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ハイクオリティな CG を利用するソフトウェアが要求する，画面描写能力の水準が上がってくるにつれて GPU の計算能力は急速に発達し，現在では単純な計算能力では CPU よりも強力な面も持つようになった．

GPGPU は発達した 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ことを目的としている．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ことに長けている．一方で，GPU が有する個々の計算機の性能は CPU よりも低く，連続的な処理は苦手としている．その為，GPU 上で実装を行うにはアルゴリズムを如何に並列化するかが重要となる．

3.2 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 向けの開発環境である [4].

CUDA ではグラフィック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，カーネルは 1 度に一つしか動作しないため，カーネル内部の並列化によりプログラムの並列化を行う．カーネルは複数のブロックを並列に動作させる．更に，ブロックは複数のスレッドを並列に動作させる．これいにより，カーネルはスレッドによる並列化が可能となる．

3.3 NVIDIA GeForce GTX 480

NVIDIA GeForce GTX 480 は 2010 年 3 月に発売された GeForce 400 シリーズの GPU である．GTX 480 は新しく Fermi アーキテクチャに搭載しており，ストリーミングマルチプロセッサ（SM）1 基当たり 32 基の計算コアを有している．更に GTX 480 は図 2 に示されるように SM を 15 基有しており，480 基の計算コアを利用することが可能である．

4 高速化手法

QUAD ストリーム暗号の高速化は多変数多項式の計算の効率化により実現することができる．GPU による実装では，計算を可能な限り

図 2: NVIDIA GeForce GTX 480 の構成

並列化することが望ましい．個々の多変数多項式は独立であるため，それぞれ並列に計算することで高速化が図れるが，それ以外の高速化手法，また使用する GPU に対する最適化を QUAD$(2, 320, 320)$ に主眼を置き本章で示す．

4.1 既存の高速化手法

Berbain らは多変数多項式暗号の実装を高速化するための高速化手法を提案している [1]．本論文では，以下の高速化手法を利用する．

第一に，MQ における二次の項 $x_i x_j$ はそれぞれ共通であるため，二次の項をあらかじめ計算し乗算回数を削減する．第二に，変数のベクトル化を行う．CUDA API の int 型は 32 ビットの変数であるため，1 ビットデータに対する 32 個の変数ベクトルとして取り扱うことができ，計算を 1/32 程度に削減可能となる．最後に，$\mathbb{GF}(2)$ 上においては，確率的に 1/2 で $x_i = 0$ であるため，確率 3/4 で $x_i x_j = 0$ となる．従って，非零な項のみを利用することで計算量を 1/4 程度に削減可能となる．

4.2 GPU 上における高速化手法

多変数二次多項式の計算では二次の項の総和が最もコストの高い部分となる．このとき，$(1 \leq i \leq j \leq n)$ となるような項 $x_i x_j$ の総和を取るため，三角行列の要素の総和を取るように考えることができる．

総和においては，順番に依らず結果を同じとなるため，先に行列の列の要素について計算し，その結果の総和をとればよい．ところで，n 次の三角行列は，1 行目に n 個，n 行目に 1 個の要素を持つ．従って，行ごとに計算を行うと，冗長な計算が増加する．n 次の三角行列は $n(n+1)/2$ 個の要素を持つため，$(n/2) \times (n+1)$，もしくは $(n+1)/2 \times n$ となるような長方形の行列に変形できる．このときの行列変形を図 3 に示す．図 3 は図 3 左の n 次の三角行列を図 3 右の長方行列に変形している．三角行列の要素和を逐次に行った場合，$n(n+1)/2$ 回の計算が必要であるのに対し，長方行列で並列的に行うことで，計算回数を $3n/2$ 回程度に収めることが可能となる．

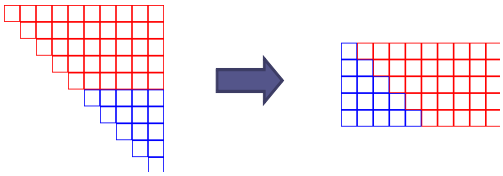

図 3: 三角行列から長方行列への変形

4.3 GPU における最適化

4.3.1 行列計算の最適化

NVIDIA GeForce GTX 480 においては 15 基の SM と SM1 基ごとに有している 32 基の計算コアを用いて並列処理を行う．各マルチプロセッサは 32 個のスレッドをワープとして一つのまとまりとして取り扱い処理を行う．従ってスレッド数が 33 といった端数が出る場合，必要なワープの数は 2 つとなり，無駄な計算コアが存在する事になる．同様に一括に制御できるマルチプロセッサの数は 15 基である．従って，マルチプロセッサで処理するスレッドの数を 32 の倍数に，1 度の処理で利用するマルチプロセッサの数を 15 の倍数にする事が望ましいといえる．

n 次の三角行列を長方行列に変形した場合，その大きさは $n(n+1)/2$ となる．この時，長辺 n 短辺 $(n+1)/2$ もしくは長辺 $n+1$ 短辺 $n/2$ となる二つのパターンが考えられる．しかしなが

ら，n によって処理を分割する事は無駄が大きい．そこである自然数 k について $n = 30k$，即ち n を 30 の倍数であると仮定する．この場合，長方行列は $15k \times (30k + 1)$ の形となる．これを拡大し，$15k \times 32k$ の長方行列として捉える．これにより，短辺をマルチプロセッサに長辺を計算コアに割り当てるような並列化が実現できる．$k = 1$ のときの例を図 4 に示す．

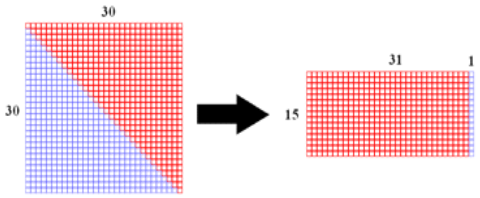

図 4: $k = 1$ であるときの $15k \times 32k$ 行列の処理

4.3.2 計算処理の最適化

GPU は並列な計算処理は得意であるが，連続な計算は苦手であり計算単位は可能な限り小規模にする必要がある．更に GPU 上の条件分岐は大きなコストとなるため，可能な限り条件分岐を削減しなければならない．

そこで GPU で動作する条件分岐を削減するために，条件によるカーネルの動作の違いをカーネルの処理内容の違いとして取り扱い，分岐を CPU 上で行う．QUAD$(2, 320, 320)$ は Berbain らの手法により，非零な項の数に応じて多項式計算における加算回数が異なる．そこで，$\mathbb{GF}(2)$ 上における非零な数毎に異なるカーネルを作成することで GPU 上での条件分岐を避けることができる．ここで，非零な変数の数を 30 ごとに分割してカウントする．QUAD$(2, 320, 320)$ においては変数の数は 320 であるため，11 個のカーネルで条件分岐可能となる．このときの非零な変数の数のカウントを k とすれば，$15k \times 32k$ の長方行列としてとらえることができる．分割を 15×32 とすれば，個々のプロセスでは $k \times k$ の小行列の総和として考えることができる．そのため，k に応じた $k \times k$ の処理を行うカーネルを作ればよい．

また $\mathbb{GF}(2)$ において非零な変数の数のカウントは CPU 上で行う．これは非零な変数の数のカウントは連続的な処理であり，並列化が非常に困難なためである．

4.4 高速化の評価

以上の高速化手法による QUAD$(2, 320, 320)$ の一回のキーストリーム生成に要する時間の変化を示す．

QUAD$(2, 320, 320)$ の一回の多項式計算における乗算回数は 51040 回必要であり，加算回数は 51359 回必要となる．多項式全体を計算するには計算回数が 640 倍となる．しかしながら，Berbain らが用いた高速化手法を利用することで一回の多項式計算における乗算回数は 12720 回，加算回数は 12880 回程度となる．更に，各多項式を独立と考え 32 ビットのベクトル変数にまとめることで多項式の計算は加算回数の 20 倍で抑えることが可能となる．

これらの手法は CPU，GPU いずれの実装でも利用可能であるが，GPU による並列化を行うと計算時間は以下の通りになる．GPU により，乗算は全て並列に処理することが可能となる．$\mathbb{GF}(2)$ 上においては $x_i x_j$ はおよそ 160×160 組程度となるため，160 のスレッドに分割することで，160 回程度の乗算時間で処理することが可能である．また，加算においては一回の多項式の計算において 240 回程度の加算を行えばよい．従って，GPU 実装は CPU 実装よりも五十分の一程度の計算回数で処理が可能となる．

更に GPU による最適化を図ることで多項式計算における加算回数は更に小さくなる．k 次の小正方行列の計算をコアすべてで行えばよいため，$k \times k$ 回の加算を行い，$32+15$ 回の加算を行えばよい．例えば，$k = 6$ の場合は $6 \times 6 = 36$ であるため，加算回数は 80 回程度となり，加算回数を更に三分の一程度に削減することができる．

5 実装実験

5.1 実験内容

QUAD$(2, 320, 320)$ について CPU，GPU それぞれで実装を行い，5MB のファイルに対する暗号化時間を測定する．GPU については最適化手法を適用した例と適用していない例について実装を行う．また，実装環境を表 1 に示す．

表 1: 実装環境

OS	Ubuntu 10.04 LTS $64bit$
CPU	Intel Core $i7$ $875K$
Memory	$8GB$ $DDR3$
GPU	NVIDIA GeForce 480 GTX

5.2 実験結果

実験結果を表 2 に示す．表 2 は CPU と GPU のそれぞれの実装に対する暗号化時間の測定結果及びスループットを表している．また，比較対象として Yang らが行った QUAD$(2, 320, 320)$ に対する実装結果 [3] も示す．GPU による実装は CPU による実装と比較して最適化前のもので 8.4 倍，最適化したものでは 29.9 倍の速度差が得られた．また，最適化前の GPU 実装の結果は Yang らのものよりも遅いものであるが，最適化により Yang らの実装よりも速い結果が得られている．しかしながら，Yang らが示す CPU の結果と比較して遅い結果となっている．

表 2: QUAD ストリーム暗号の実装結果

		暗号化時間	スループット
CPU		$326.52s$	$0.13Mbps$
GPU(非最適化)		$38.96s$	$1.03Mbps$
GPU(最適化)		$10.94s$	$3.66Mbps$
Yang et al. [3]	CPU		$6.10Mbps$
	GPU		$2.60Mbps$

6 まとめ

QUAD ストリーム暗号の高速化を図るため，GPU を用いた実装を行った．GPU 実装と最適化を用いることで CPU による実装の約 30 倍の速度結果が得られた．これにより，Yang らの示す GPU の実装よりも早い結果を得る事ができたが，Yang らの示す CPU による結果よりは遅いものであった．これは GPU による実装の最適化にまだ幾らかの改善の余地があるものと考える．更なる最適化を図り，QUAD ストリーム暗号の実用性を上げる事が今後の課題である．

参考文献

[1] Berbain, C., Billet, O., Gilbert H.: Efficient Implementations of Multivariate Quadratic Systems. In: The 13th Annual Workshop on Selected Areas in Cryptography. LNCS, vol. 4004, pp. 174-187. Springer-Verlag (2006)

[2] Berbain, C., Gilbert, H., Patarin, P.: QUAD: a Practical Stream Cipher with Provable Security. In: EUROCRYPT 2006. LNCS, vol. 4004, pp. 109-128. Springer-Verlag (2006)

[3] Chen, M.-S., Chen, T.-R., Cheng, C.-M., Hsiao, C.-H., Niederhagen R., Yan, B.-Y.: What price a provably secure stream cipher? In: Fast Software Encryption, 2010, Rump session. (2010)

[4] NVIDIA CUDA `http://developer.NVIDIA.com/object/CUDA.html`

[5] Patarin, J., Goubin, L.: Asymmetric cryptography with s-boxes. In: Information and Communication Security, First International Conference. pp. 369-380, (1997).

[6] Yang, B.-Y., Chen, O. C.-H., Bernstein, D. J., Chen, J.-M.: Analysis of QUAD. In: Fast Software Encryption 2007. LNCS, vol. 4593, pp. 290-308. Springer-Verlag (2007).