Accelerating of Solving Method for Non-linear Multivariate System with Graphics Processing Unit

Satoshi Tanaka* Bo-Yin Yang** Chen-Mou Cheng*** Kouichi Sakurai****
(*Graduate school, ****Faculty of Information Science and Electrical Engineering, Kyushu University, Japan)
(****Academia Sinica, Taiwan) (****National Taiwan)
{tanasato@itslab.inf, sakurai@csce.kyushu-u.ac.jp, by@crypto.tw, ccheng@cc.ee.ntu.edu.tw}

1 Introduction
The security of public-key cryptosystems depends on the complexity of math problems. Multivariate public-key cryptosystems (MPKC) are expected to secure against quantum computers. The security of MPKC is based on the complexity of solving non-linear multivariate polynomial equations over a finite field (MP). MP is an NP-complete problem.

In this paper, we show implementations of the XL algorithm on GPU (Graphics Processing Units). We try to accelerate the XL algorithm and compare with CPU implementations.

2 XL Algorithm

2.1 MP problem
Let x be an n-vector of $F_q$, where $q = p^k$, $p$ is a prime and $k \leq 1$, and $f(x)$ be a multivariate polynomial function. The MP problem is solving non-linear polynomial equations by given $y = \{f_1(x), \ldots, f_m(x)\}$. It is known to an NP-complete problem.

2.2 XL Algorithm
The XL algorithm was proposed by Courtois in 2000. It constructs a new multivariate polynomial. We describe the XL algorithm[2] in Algorithm 1.

Algorithm 1 The XL algorithm[2]
1: Multiply: Generate all the product of polynomial equations and products of unknowns $\prod_{j=1}^{m} x_{i_j}$.
2: Linearize: Consider each monomial in the $x_i$ of degree $\leq D$ as a new unknown and perform an elimination algorithm on the equations obtained in 1 and derive univariate equations.
3: Solve: Solve univariate equations obtained in 2 over $F_q$.
4: Repeat: Simplify the equations and repeat the process to find the values of other unknowns.

2.3 XL on GPU
The GPU version of the XL algorithm needs parallelization. We parallelize each step of Algorithm 1. In “multiply” and “repeat” steps, we parallelize generating and simplifying for each equation respectively. Moreover, we also parallelize for each combination of products $\prod_{j=1}^{m} x_{i_j}$. In “linearize” step, we use Gaussian elimination for elimination algorithm. Since each row of matrix is independently of other rows, we can parallelize row additions. We also parallelize computing each element of row vectors. In “solve” step, we use the brute force search in parallel.

Table 1: The results of solving time for quadratic polynomial equations

<table>
<thead>
<tr>
<th>Unknowns</th>
<th>Polynomials</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>16</td>
</tr>
<tr>
<td>GPU</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>30</td>
</tr>
</tbody>
</table>

3 Experimentation
We implement the XL algorithm on CPU and GPU, and measure solving time random quadratic polynomial equations with n unknowns, where $1 \leq n \leq 16$.
All the experiments are performed on Ubuntu 10.04 LTS 64bit, Intel Core i7 875K and NVIDIA GeForce 580 GTX. Moreover, we use 8 GBytes of DDR3 memory.

3.1 Result
We present the results of solving time for quadratic polynomial equations, where $12 \leq n \leq 16$ in Table 1.

4 Summary
In this paper, we describe the XL algorithms on GPU and compare with CPU implementations. GPU version can solve systems with 16 unknowns and 40 quadratic equations over $F_2$ in about 11 seconds. It is 100 times faster than CPU version.

5 Further work
We would like to implement the block Wiedemann algorithm[1] on GPU. The XL algorithm constructs a sparse matrix on the multiply step. The block Wiedemann algorithm makes more efficient on the linearize step than Gaussian elimination.

Acknowledgement
The first author’s visiting research in National Taiwan University is supported by Graduate School of Information Science and Electrical Engineering, Kyushu University.

References