Fast Parallel Garner Algorithm for Chinese Remainder Theorem

Yongnan Li, Limin Xiao, Aihua Liang, Yao Zheng, and Li Ruan

School of Computer Science and Engineering, Beihang University,
Beijing, 100191, China
{liyongnan1984,liangah,zyshren}@cse.buaa.edu.cn,
{xiaolm,ruanli}@buaa.edu.cn

Abstract. This paper presents a fast parallel garner algorithm for Chinese remainder theorem. The variables in garner algorithm are divided into public parameters that are constants for fixed module and private parameters that represent random input integers. We design the parallel garner algorithm by analyzing the data dependencies of these arithmetic operations for computing public variables and private variables. Time complexities and speedup ratios of the parallel algorithm and the sequential algorithm are calculated to make the quantitative comparison based on our previous work about some fundamental parallel algorithms. The performance evaluation shows high efficiency of the proposed parallel algorithm compared to the sequential one.

Keywords: garner algorithm, Chinese remainder theorem, parallel processing, balanced binary tree.

1 Introduction

The Chinese remainder theorem [1], which is creatively put forward by Sun Tzu, an ancient Chinese military strategist who composed the brilliant military writing "The Art of War", is a constructive algorithm to find the solution of a positive integer divided by some given divisors. In recent years, this theorem has received considerable attention in many modern computer applications, especially in the field of information security. There are many scientists dedicated to simplifying and accelerating the operation of Chinese remainder theorem to reduce the computation complexities in these applications. These works about parallelization of Chinese remainder theorem are mainly focused on fault-tolerant technology [2,3], binary reverse converter [4-6], distributed key distribution scheme [7,8] and fast encryption/decryption of cryptographic algorithm [9-11]. However, there is less deep study on more general parallel algorithms concerning multiple-precision integers for Chinese remainder theorem.

In this paper, we propose a fast parallel garner algorithm concerning basic algebra and modular arithmetic operations of multiple-precision integer to increase the efficiency of Chinese remainder theorem. The general parallel methods including balanced binary tree and prefix computation circuit are chosen to design the proposed parallel algorithm based on analyzing the data dependencies of the mathematical

J.J. Park et al. (Eds.): NPC 2012, LNCS 7513, pp. 164–171, 2012.

operations. This parallel algorithm achieves high speedup and could be applied to many types of applications by improving their efficiencies.

The rest of this paper is organized as follows. Next section introduces the garner algorithm and section 3 presents time complexities of several basic arithmetic algorithms. Section 4 proposes the fast parallel garner algorithm. The performance evaluation is presented in section 5. The last section concludes the whole paper and points out some future works briefly.

2 Garner Algorithm

The following algorithm is the garner algorithm, by means of which can be found the solution of variables divided by the given divisors in Chinese remainder theorem. For more details about Chinese remainder theorem, please refer to [1]. Product *M* of the dividers m_i represents the module in Chinese remainder theorem while sequences $(v_1, v_2, \ldots, v_t)$ map given remainders in different finite field m_i.

Garner Algorithm

Input: positive integer $M = \prod_{i=1}^{t} m_i > 1$, for $\forall i \neq j, gcd(m_i, m_j) = 1$,
$v(x) = (v_1, v_2, \ldots, v_t)$.

Output: integer *x*.

1. for *i* from 2 to *t*, repeat:
   1.1 $C_i \leftarrow 1$.
   1.2 for *j* from 1 to $(i-1)$, repeat:
   $u_j \leftarrow m_j^{-1} mod\ m_i, \quad u_j \leftarrow u_j \bullet C_j mod\ m_i$.
2. $u \leftarrow v_1,\ x \leftarrow v_1$.
3. for *i* from 2 to *t*, repeat:
   $u \leftarrow (v_i - x)C_i mod\ m_i, \quad x \leftarrow x + u \bullet \prod_{j=1}^{i-1} m_j$.
4. return (*x*).

3 Time Complexities of Basic Algorithms

As depicted in [12], time complexities of basic multiple-precision algorithms are listed in Table 1. We set the runtime of single-precision multiplication as the basic measurement unit. Time complexities of multiple-precision addition and deduction are *O(1)*.

Table 1. Time complexities of basic algorithms

Operation	Parallel	Sequential
Multiplication	$\overbrace{O(n^2/s+2)}^{computation} + \overbrace{O(n^2/s+2n)}^{communication}$	$O(n^2+2n)$
Barrett reduction	$\overbrace{O(2n^2/s+8)}^{computation} + \overbrace{O(2n^2/s+4n)}^{communication}$	$O(n^2+4n+5)$
Inversion-multiplication	$\overbrace{O(3\lceil \lg X \rceil + 3\lceil \lg P \rceil)}^{computation} + \overbrace{O(2\lceil \lg X \rceil + 2\lceil \lg P \rceil)}^{communication}$	$O(4\lceil \lg X \rceil + 4\lceil \lg P \rceil)$

The meanings of the variables in Table 1 list as follows:

- n : multiple-precision of operand.
- s : process number for computing multiplication.
- X: numerator of inversion-multiplication.
- P: denominator of inversion-multiplication.

4 Parallel Garner Algorithm

This section discusses the proposed parallel garner algorithm for Chinese remainder theorem. This algorithm contains three public parameters (C_i, $m_j^{-1} mod\ m_i$ and $\prod_{j=1}^{i-1} m_j$) and two private parameters (v_i and x). The public parameters are fixed integers and only need to be computed one time for the same module *M*, while the private ones are used to compute the random number *x* divided by the given divisors.

4.1 Parallelization of Inversion

Inversion $m_j^{-1} mod\ m_i$ is one particular case of inversion-multiplication, in which the dividend is integer 1, and the parallel complexity and sequential complexity could be looked up in Table 1. For every m_i in substep 1.2, $(i-1)i/2$ times inversion operations should be calculated. These inversion operations have no data dependency, which means that all of them could be computed simultaneously. Therefore, the parallel runtime and sequential runtime are

$$T_p = \overbrace{O(3\lceil lg\ X\rceil + 3\lceil lg\ P\rceil)}^{computation} + \overbrace{O(2\lceil lg\ X\rceil + 2\lceil lg\ P\rceil)}^{communication}. \tag{1}$$

$$T_s = (t(t-1)(t+1)/6)O(4\lceil lg\ X\rceil + 4\lceil lg\ P\rceil). \tag{2}$$

Assume that the word length of the computer we used is *k* bit, and then total runtime lists as follows

$$T_{p1} = \overbrace{O(6kn_{max})}^{computation} + \overbrace{O(4kn_{max})}^{communication}\ where\ n_{max} = \{n_1, n_2, \ldots, n_{t-1}\}. \tag{3}$$

$$T_{s1} = \sum_{i=2}^{t}(i-1)O(8kn_i). \tag{4}$$

4.2 Parallelization of Computing C_i

In substep 1.2, we could obtain the solution of C_i by computing $\prod_{j=1}^{i-1} u_j mod\ m_i$. Then computing C_i is a classical balanced binary tree problem, which is one of the general parallel questions. Fig.1 shows one example of computing C_i by using balanced binary tree. In every layer of the balanced tree, one multiplication and one reduction must be executed. The depth of the balanced tree is $\lceil lg(i-1)\rceil$, so the parallel runtime for computing C_i is

$$T_p = \lceil lg(i-1) \rceil \Big(\overbrace{O(3n_i^2/s + 10)}^{computation} + \overbrace{O(3n_i^2/s + 6n_i)}^{communication} \Big). \quad (5)$$

All of public parameters C_i have no data dependency, so they can be parallel calculated. Then the parallel runtime for computing all C_i is

$$T_{p2} = Max \left\{ \lceil lg(i-1) \rceil \left(\overbrace{O(3n_i^2/s + 10)}^{computation} + \overbrace{O(3n_i^2/s + 6n_i)}^{communication} \right) \right\}. \quad (6)$$

In sequential algorithm, $(i-1)$ times multiplications and $(i-1)$ times reductions are needed. The sequential runtime is

$$T_s = O(\textstyle\sum_{j=1}^{i-1}(2n_j^2 + 6n_j + 5)). \quad (7)$$

Therefore, the sequential runtime for computing all C_i is

$$T_{s2} = O(\textstyle\sum_{i=2}^{t} \sum_{j=1}^{i-1}(2n_j^2 + 6n_j + 5)). \quad (8)$$

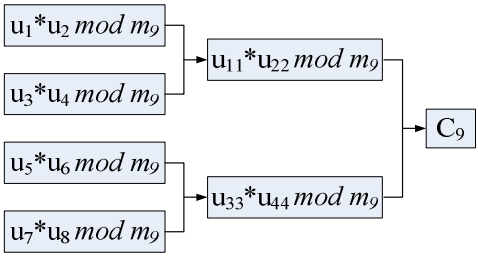

Fig. 1. An example of computing C_i

4.3 Parallelization of Product of Subset Module

M_i is defined as $\prod_{j=1}^{i} m_j$ for clarifying expression of the parallel procedure of this series of multiplication operation. We adopt the high-low prefix computation circuit, a general parallel method to handle suffix computation problem, to calculate this operation and Fig.2 shows one example of computing $\{M_1, M_2, \ldots, M_{t-1}\}$.

As depicted in Fig.2, $\lceil lg(t-1) \rceil$ communication time units and $\lceil lg(t-1) \rceil$ round multiplications are required when computing$\{M_1, M_2, \ldots, M_{t-1}\}$. Along with the execution of the high-low prefix computation circuit, the multiple-precision of the parameters are doubled. Therefore, the parallel runtime for computing $\{M_1, M_2, \ldots, M_{t-1}\}$ is

$$T_{p3} = \textstyle\sum_{i=1}^{\lceil lg(t-1) \rceil} \left\{ \overbrace{O((in_{max})^2/s + 2)}^{computation} + \overbrace{O((in_{max})^2/s + 2in_{max} + 1)}^{communication} \right\}. \quad (9)$$
$$where\ n_{max} = max\{n_1, n_2, \ldots, n_{t-1}\}$$

If the multiple-precisions of two parameters in multiplication are l_1 and l_2 respectively, the sequential time complexity would be $O(l_1 l_2 + l_1 + l_2)$. In sequential

algorithm of computing $M_i = M_{i-1} \bullet m_i$, the multiple-precision of M_{i-1} is $\sum_{j=1}^{i-1} n_j$ and the one of m_i is n_i. Therefore, the sequential runtime of computing $M_{i-1} \bullet m_i$ is

$$T_s = O\Big((\sum_{j=1}^{i-1} n_j)n_i + \sum_{j=1}^{i} n_j\Big). \tag{10}$$

Then the total sequential runtime for computing $\{M_1, M_2, \ldots, M_{t-1}\}$ is

$$T_{s3} = O\left(\sum_{i=2}^{t} \left((\sum_{j=1}^{i-1} n_j)n_i + \sum_{j=1}^{i} n_j\right)\right). \tag{11}$$

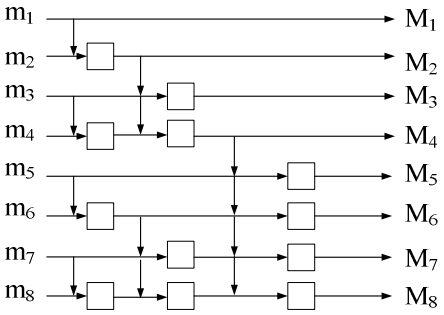

Fig. 2. An example of computing M_{t-1}

4.4 Parallelization of Computing Private Variables

The private variables in garner algorithm concerns only two operations in step 3. Degrading the multiple-precision of x is helpful to simplify the computation of $(v_i - x)C_i \ mod \ m_i$, so one reduction needs to be executed firstly. If v_i is smaller than $x \ mod \ m_i$, one multiple-precision addition would be needed and the probability is 0.5. To sum up, this operation consists of 0.5 multiple-precision addition, one multiple-precision deduction, one multiplication and two reductions. Then the parallel runtime and sequential runtime of computing $(v_i - x)C_i \ mod \ m_i$ are

$$T_p = \overbrace{O(5n_i^2/s + 19.5)}^{computation} + \overbrace{O(5n_i^2/s + 10n_i)}^{communication}. \tag{12}$$

$$T_s = O(3n_i^2 + 10n_i + 11.5). \tag{13}$$

Therefore, the total runtime of this operation for all rounds in this loop are

$$T_{p4} = \sum_{i=2}^{t} \left(\overbrace{O(5n_i^2/s + 19.5)}^{computation} + \overbrace{O(5n_i^2/s + 10n_i)}^{communication}\right). \tag{14}$$

$$T_{s4} = \sum_{i=2}^{t} O(3n_i^2 + 10n_i + 11.5). \tag{15}$$

In every round of the loop in step 3, the multiple-precision of u and M_{i-1} are n_i and $\sum_{j=1}^{i-1} n_j$ respectively. The operation $(x + u \bullet M_{i-1})$ contains one multiplication and

one multiple-precision addition. Therefore, the parallel runtime and sequential runtime of every round are

$$T_p = \overbrace{O\Big(n_i\Big(\sum_{j=1}^{i-1} n_j\Big)/s+3\Big)}^{computation} + \overbrace{O\Big(n_i\Big(\sum_{j=1}^{i-1} n_j\Big)/s+\sum_{j=1}^{i} n_j\Big)}^{communication}. \quad (16)$$

$$T_s = O\Big(n_i(\sum_{j=1}^{i-1} n_j)+\sum_{j=1}^{i} n_j+1\Big). \quad (17)$$

The total runtime of this operation are

$$T_{p5} = \sum_{i=2}^{t} \overbrace{O\Big(n_i\Big(\sum_{j=1}^{i-1} n_j\Big)/s+3\Big)}^{computation} + \overbrace{O\Big(n_i\Big(\sum_{j=1}^{i-1} n_j\Big)/s+\sum_{j=1}^{i} n_j\Big)}^{communication}. \quad (18)$$

$$T_{s5} = \sum_{i=2}^{t} O\Big(n_i(\sum_{j=1}^{i-1} n_j)+\sum_{j=1}^{i} n_j+1\Big). \quad (19)$$

5 Performance Discussion

This section evaluates the performance of the parallel garner algorithm and sequential garner algorithm. In order to analyze the performance of this algorithm, we choose the special remainder module in which all given dividers have the same multiple-precision in garner algorithm. In other words, all n_i are equal in this algorithm and we assign value n_a to it. For simplification of performance evaluation, we also assume that the value of s is n_a and that the word length of the computer is 32. Then parallel runtime of all operations could be simplified as the following expressions with only two variables:

$$T_{p1} = \overbrace{O(192n_a)}^{computation} + \overbrace{O(128n_a)}^{communication}. \quad (20)$$

$$T_{p2} = \lceil lg(t-1) \rceil \left(\overbrace{O(3n_a+10)}^{computation} + \overbrace{O(9n_a)}^{communication} \right). \quad (21)$$

$$T_{p3} = \sum_{i=1}^{\lceil lg(t-1) \rceil} \left(\overbrace{O(i^2 n_a+2)}^{computation} + \overbrace{O(i^2 n_a+2in_a+1)}^{communication} \right). \quad (22)$$

$$T_{p4} = (t-1)\left(\overbrace{O(5n_a+19.5)}^{computation} + \overbrace{O(15n_a)}^{communication} \right). \quad (23)$$

$$T_{p5} = \overbrace{O((t-1)tn_a/2+3(t-1))}^{computation} + \overbrace{O((t^2-1)n_a)}^{communication}. \quad (24)$$

And so are the simplifications of the sequential runtime of all operations:

$$T_{s1} = O\big(128t(t-1)n_a\big). \quad (25)$$

$$T_{s2} = O\big(t(t-1)(2n_a^2+6n_a+5)/2\big). \quad (26)$$

$$T_{s3} = O\big((t-1)tn_a^2/2+(t-1)(t+2)n_a/2\big). \quad (27)$$

$$T_{s4} = O\big((t-1)(3n_a^2 + 10n_a + 11.5)\big). \tag{28}$$

$$T_{s5} = O\big((t-1)tn_a^2 + (t-1)(t+2)n_a/2 + t - 1\big). \tag{29}$$

The speedup is

$$S = (T_{s1} + T_{s2} + T_{s3} + T_{s4} + T_{s5})/(T_{p1} + T_{p2} + T_{p3} + T_{p4} + T_{p5}). \tag{30}$$

We assign the independent parameters (n_a and t) into different values to make the quantitative comparison between the parallel algorithm and the sequential algorithm on condition that the communication time unit is 20 percent of computation time unit. As showed in Fig.3, the proposed method could significantly accelerate the speed of incorporating the parameters in the subset of Chinese remainder theorem. The quantitative performance evaluation demonstrates acceleration up to 9~55 times speedup while the parameter t varies from 5 to 11 and parameter n_a increases from 5 to 40. We also make other assumption of relationship between communication time unit and computation time unit. The same conclusion could be derived by analyzing the performance comparison on different conditions.

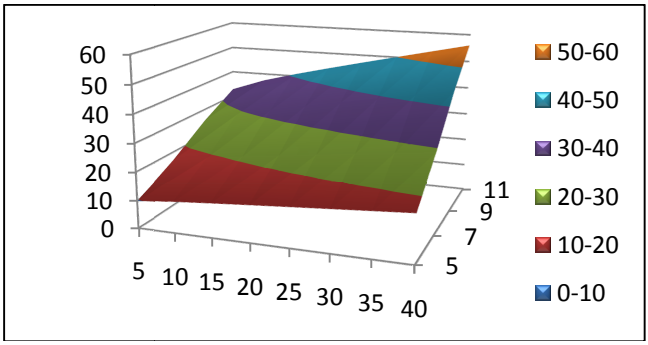

Fig. 3. Speedup ratio of parallel garner algorithm

6 Conclusions

This paper proposes a fast parallel garner algorithm designed for Chinese remainder theorem, which concerns basic algebra and modular arithmetic operations of multiple-precision integer, for rapid calculation of a random number divided by some given divisors. The parallel algorithm is designed through a separate consideration of the constants for given module that is called public variables and random given parameters named as private variables. The public variables only need to be computed one time for the same module. Our previous work about time complexities of some basic operations serves as a simple and convenient criterion for the proposed algorithm. The performance evaluation and the comparison demonstrate that the fast parallel garner algorithm achieves remarkable speedup.

The analysis of this parallel algorithm is only a theoretical one and it just make performance comparison by considering different multiple precision integers of the variables in garner algorithm. We are likely to implement the parallel garner algorithm on GPU to verify the validity and high efficiency of this algorithm. Future research efforts may also focus on the application of this fast parallel garner algorithm in the field of

fault-tolerant technology, binary reverse converter, distributed key distribution scheme, fast encryption/decryption, etc.

Acknowledgments. This study is sponsored by the National "Core electronic devices high-end general purpose chips and fundamental software" project under Grant No. 2010ZX01036-001-001, Beijing Natural Science Foundation under Grant No. 4122042, the Hi-tech Research and Development Program of China (863 Program) under Grant No. 2011AA01A205 and the National Natural Science Foundation of China under Grant No. 60973008.

References

1. Wikipedia, http://en.wikipedia.org/wiki/Chinese_remainder_theorem
2. Chen, H.: CRT-based high-speed parallel architecture for long BCH encoding. IEEE Transactions on Circuits and Systems II: Express Briefs 56(8), 684–686 (2009)
3. Sundaram, S., Hadjicostis, C.N.: Fault-tolerant convolution via Chinese remainder codes constructed from non-coprime moduli. IEEE Transactions on Signal Processing 56(9), 4244–4254 (2008)
4. Abdelfattah, O., Swidan, A., Zilic, Z.: Direct residue-to-analog conversion scheme based on Chinese remainder theorem. In: 2010 IEEE International Conference on Electronics, Circuits, and Systems, pp. 687–690. IEEE Press, Athens (2010)
5. Hariri, A., Navi, K., Rastegar, R.: A new high dynamic range moduli set with efficient reverse converter. Computers and Mathematics with Applications 55(4), 660–668 (2008)
6. Bi, S., Gross, W.J.: The mixed-radix Chinese remainder theorem and its applications to residue comparison. IEEE Transactions on Computers 57(12), 1624–1632 (2008)
7. Zhou, J., Ou, Y.-h.: Key Tree and Chinese Remainder Theorem Based Group Key Distribution Scheme. In: Hua, A., Chang, S.-L. (eds.) ICA3PP 2009. LNCS, vol. 5574, pp. 254–265. Springer, Heidelberg (2009)
8. Wen, T.Z.: Analyzing euler-fermat theorem based multicast key distribution schemes with Chinese remainder theorem. In: 2008 IFIP International Conference on Network and Parallel Computing, pp. 11–17. IEEE Press, Shanghai (2008)
9. Song, B., Ito, Y., Nakano, K.: CRT-based DSP decryption using montgomery modular multiplication on the FPGA. In: 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum, pp. 532–541. IEEE Press, Anchorage (2011)
10. Li, Y., Xiao, L., Chen, S., Tian, H., Ruan, L., Yu, B.: Parallel Extended Basic Operations for Conic Curves Cryptography over Ring Zn. In: 9th IEEE International Symposium on Parallel and Distributed Processing with Applications Workshops, pp. 203–209. IEEE Press, Busan (2011)
11. Li, Y., Xiao, L., Qin, G., Li, X., Lei, S.: Comparison of Three Parallel Point-Multiplication Algorithms on Conic Curves. In: Xiang, Y., Cuzzocrea, A., Hobbs, M., Zhou, W. (eds.) ICA3PP 2011, Part II. LNCS, vol. 7017, pp. 43–53. Springer, Heidelberg (2011)
12. Li, Y., Xiao, L., Hu, Y., Liang, A., Tian, L.: Parallel algorithms for cryptosystem on conic curves over finite field Fp. In: 9th International Conference on Grid and Cloud Computing, pp. 163–167. IEEE Press, Nanjing (2010)