Parallel Strategy to Factorize Fermat Numbers with Implementation in Maple Software

Jianhui Li*, Manlan Liu
Foshan Polytechnic, Foshan City, PRC, 528000, China.

* Corresponding author. Tel.: +86075787263015; email: joe863@163.com
Manuscript submitted October 08, 2020; accepted December 11, 2020.
doi: 10.17706/jsw.16.4.167-173

Abstract: In accordance with the traits of parallel computing, the paper proposes a parallel algorithm to factorize the Fermat numbers through parallelization of a sequential algorithm. The kernel work to parallelize a sequential algorithm is presented by subdividing the computing interval into subintervals that are assigned to the parallel processes to perform the parallel computing. Maple experiments show that the parallelization increases the computational efficiency of factoring the Fermat numbers, especially to the Fermat number with big divisors.

Key words: Integer factorization, fermat number, parallel computing, algorithm.

1. Introduction

Factorization of big integer has been a hard problem and has been paid attention to in mathematics and cryptography. Any factoring algorithm is both scientifically valuable and practically applicable, as stated in [1]. Historically, factorization of big integer always took huge computing resources and involved in parallel computing. For example, it took half a year and 80 4-cored CPUs of AMD Opteron @ 2.2GHz to factorize RSA768 [2]. Actually, Brent R P began to study the possibility of parallel computing early in 1990 since Pollard Rho algorithm came into being [3]. And then parallel approaches for the elliptic curve method (ECM), continued fraction, quadric sieve (QS) and number field sieve (NFS) were developed respectively [4]-[7]. Now the NFS has been regarded to be the most efficient method. However, the method is popularly regarded to consume huge computing resource in the factorization. Anuja A Sonalker showed in his master thesis [8], it would need 240M memory for factor base, 10GB memory for sieve and 160G for matrix to factorize an objective of 768 binary digits, and the number of memory will be 7.5G, 256GB and 10T for an objective of 1024 binary digits. It was reported that, Bruce Schneier spent 4000 core-years, using Intel Xeon Gold 6130 CPUs as a reference (2.1GHz), on factoring the RSA240.

The Fermat number $F_m = 2^{2^m} + 1$ is a kind of eyeball catching numbers. It becomes very big with the increment of m. When $m > 10$, its number of binary digits is over 1024. Accordingly, factorization of the Fermat numbers has been focused in computing science. Nowadays, grid computing plays the major role in factoring the kind of numbers. Volunteers all over the world contribute the work, as seen in the website hosted and maintained by professor Wilfrid Keller [9], [10].

A recent paper [11] published on Journal of Software (JSW) put forward an algorithm to factorize the Fermat numbers. The algorithm was said to be a new one and available for factoring big Fermat number F_m ($m > 100001$). The paper [11] list mere a sequential algorithm with experiments in Maple software. With a detail analysis on the algorithm and based on our experience in parallel computing [12], we found the algorithm could be parallelized. This paper accordingly gives a parallel algorithm.

2. Foundation of Algorithm

2.1. Analysis of the Algorithm

According to proposition 2 in [11]，for an odd integer $N = pq$ with divisors $q = 2^{\alpha} u + 1$ and p satisfying $2^{\alpha+\beta} + 1 \le p \le 2^{\alpha+\beta+1} - 1$, it holds

$$\left\lfloor \frac{N-1}{2^{2\alpha+\beta+1}} \right\rfloor - 1 \le u \le \left\lfloor \frac{N-1}{2^{2\alpha+\beta}} \right\rfloor$$

$$2\alpha + \beta + 1 \le \lfloor \log_2 N \rfloor$$

$$\alpha \le \left\lfloor \frac{\log_2 N}{2} \right\rfloor - 2$$

where α and β are positive integers, $u > 1$ is an odd integer.

Let

$$n_u = \frac{\left\lfloor \frac{N-1}{2^{2\alpha+\beta}} \right\rfloor - (\left\lfloor \frac{N-1}{2^{2\alpha+\beta+1}} \right\rfloor - 1)}{2} + 1$$

then

$$\left\lfloor \frac{u}{4} \right\rfloor + 1 \le n_u \le \left\lfloor \frac{u}{2} \right\rfloor + 3$$

Since the Fermat numbers are of the form $F_m = 2^{2^m} + 1$ and have divisors of the form $2^n u + 1$ with $n \ge m + 2$, a searching algorithm can be designed to find out u, and then find out a divisor of F_m by computing the greatest common divisor between F_m and $2^j u + 1$ （$1 \le j \le \alpha$）. Article [11] accordingly designed an algorithm that can find a divisor within $O(0.25u(\log_2 N)^2)$ searching steps.

It can see that, the designed algorithm is suitable and efficient in sequential computing for the cases when u is small. For a big u, for example, one u in F7 is 116503103764643, will take a long searching time. With the development of multi-core computers, parallel computing is going to be popular. With a parallel computing, it is sure to decrease the searching time. Since a parallel computing requires a subdivision of the computing task into small scale, we introduce a plan for the parallel factorization of the Fermat number next.

2.2. Subdivision of U Interval

Suppose u is in an odd interval U (odd interval: an interval [a, b] contains odd integers with odd integers a and b being respectively the lower bound and upper bound); let L be the length of U, namely, the number of odd integers contained in the interval, $m = n_p + 1$ be the number of total processes taking a part into the parallel computing ; then there are integers $s \ge 0$ and $0 \le r < n_p$, such that

$$L = sn_p + r$$

If $s = 0$, it means the number of odd integers (terms) contained in U is no more than the number of total processes in computing, and thus it is not necessary to apply parallel computing. Accordingly, we next assume $s>1$ and subdivide U into m subintervals, among which n_p contains s terms and another one contains r terms. For convenience, let $I_0, I_1, \ldots, I_{m-2}, I_{m-1}$ be the subdivided m subintervals, among which I_0 contains r terms and each of the other ones contains s terms. Assume I_0 is the leftmost one, the subdivision is illustrated in Fig. 1.

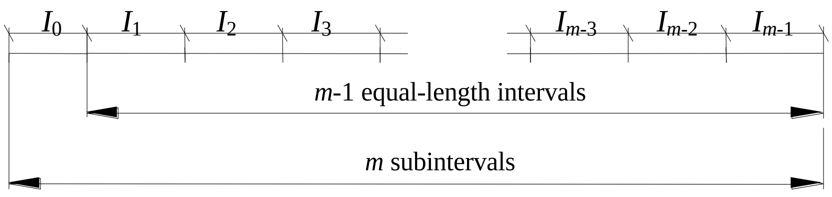

Fig. 1. U interval subdivision.

Let $U=[u_l,u_r]$, $u_0=u_l, u_1=u_0+2r$; then

$$I_0=[u_0,u_0+2(r-1)],$$
$$I_j=[u_1+2(j-1)s, u_1+2js-2],\ j=1,2,\ldots,m-1$$

form the subdivision of the U interval. For example, $I_u=[13,169]$, $m=8=7+1$; then $L=\frac{169-13}{2}+1=79$ and thus $s=11, r=2$, $u_0=13$, $u_1=13+2\times 2=17$. The subdivision of I_u is as follows.

$$I_0=[13,15]$$
$$I_1=[17,17+2\times1\times11-2]=[17,37]$$
$$I_2=[17+2\times1\times11,17+2\times2\times11-2]=[39,59]$$
$$I_3=[17+2\times2\times11,17+2\times3\times11-2]=[61,81]$$
$$I_4=[17+2\times3\times11,17+2\times4\times11-2]=[83,103]$$
$$I_5=[17+2\times4\times11,17+2\times5\times11-2]=[105,125]$$
$$I_6=[17+2\times5\times11,17+2\times6\times11-2]=[127,147]$$
$$I_7=[17+2\times6\times11,17+2\times7\times11-2]=[149,169]$$

3. Algorithm Design and Experiments

This section presents a parallel algorithm in accordance with the analysis and the U interval subdivision plan stated in previous section. Experiments in Maple software are also shown.

3.1. The Algorithm

Subdivision of the U interval lays a foundation for the parallel computing. Let $m=n_p+1$ be the number of total processes that take a part into the computation, and the U interval be subdivided as planed, then each subinterval is assigned to a process to realize the parallel computing. By this means, there must be a process that can reach the goal of the computing. The following Algorithm 1 is the designed parallel algorithm. In the algorithm, the procedure OnInterval is executed on each subinterval by a process to find the greatest common divisor between N and each term contained in the subinterval. The input parameter n means the index of the Fermat number F_n, is the same as that in [11].

```
Algorithm 1: Parallel Algorithm to Facorize F_n
1: Inputs: n, $\chi$ ;
2: Begin
3: Calculte  $b=2^n-\chi$;  $a=n+13$;  $N=2^{2^n}+1$;
4: P= the number of total process to join the computation;
5: for i from b downto 3 do
6:      for j from n+2 to a do
7:      Compute $u_l=2^{2^n-i}-1, u_r=2^{2^n-i+1}+1, L=2^{2^n-j-1}+2$;
8: Compute  $s=\left\lfloor \frac{L}{P-1} \right\rfloor, r=L-s(P-1)$ ;
9: Initialize interval U:  $u_0=u_l, u_1=u_0+2r$ ;
10: Host process perform OnInterval on  $[u_0, u_1-2]$; if success then
       stop the whole computation;
11: for k from 1 to P-1 do
12: Process k perform OnInterval on  $[u_1+2(k-1)s, u_1+2ks-2]$ ;
        if successes then tells the other process to stop and then stop
        itself;
13: end for j;
14: end for i
15: END proc
```

```
Procedure OnInterval
1: Inputs: N,j, ul,ur; # searching inteval [ul,ur]
2: for u from ul to ur step 2 do
3:    Compute g=gcd(N,2^j u+1);
4:    if (g > 1) then return g;end if
5: end do u
6: return 1;
7: END proc
```

3.2. Maple Experiments

Based on the previous parallel algorithm, experiments were made on a personal computer with 8-cored CPU of E5410 @2.33GHz, 8G memories. We chose Maple-18 software as the software platform. With Maple script programs and 7 cores, the parallel experiments were established to find the small divisor of the Fermat numbers. The results are list in Table 1. In the table, the values of the parameter are taken from [11], the column 'Core ID' means the coding number of the process that find the computing result, the columns 'Parallel searching steps' and 'Sequential searching steps' are computing steps respectively with parallel and sequential computing to find out the result, the column 'Promoting ratio' is calculated by

$$\text{Promoting ratio} = \frac{\text{Parallel searching Steps}}{\text{Sequentialsearching Steps}}$$

It is mandatory to point out that, due to the limitation of memory in the given computing environment, Maple-18 is not able to compute F30 and a larger Fermat number in sequential computing, neither to compute F29 because it has 'not enough memory to allocate '. Seen from the results in Table 1, the computational efficiency was promoted a lot when factoring a Fermat number that has a big 'small divisor', namely, the small divisor of the Fermat number is big. For example, in finding the divisor factoring 31065037602817 of F17, the sequential computing takes 12848717 steps while the parallel computing takes 1663905 steps. The promoting ratio is 7.7. The promoting ratio of the whole computing is shown with Fig. 2.

Table 1. Comparison of Sequential Computation vs. Parallel Computation

Fermat Numers		Found Divisor	Core ID	Parallel searching Steps	Sequential searching Steps	Promoting ratio
F5	2	641	0	5	5	1
F6	10	274177	1	28	28	1
F9	5	2424833	2	24	102	4.25
F10	13	45592577	3	108	1474	13.6
F11	5	319489	2	5	8	1.6
F12	2	114689	1	5	6	1.2
F13	25	2710954639361	2	3905732	20682950	5.3
F15	9	1214251009	1	214	1074	5.0
F16	10	825753601	4	111	795	5.30
F17	25	31065037602817	5	1663905	12848717	7.7
F18	3	13631489	4	4	7	1.75
F19	15	70525124609	1	435	435	1
F21	19	4485296422913	1	5205	5205	1
F23	2	167772161	0	5	6	1.2
F25	15	25991531462657	3	7829	40601	5.2
F26	17	76861124116481	1	16975	71590	4.2
F27	17	151413703311361	1	15900	70515	4.4
F29	20	2405286912458753	/	/	35741	/

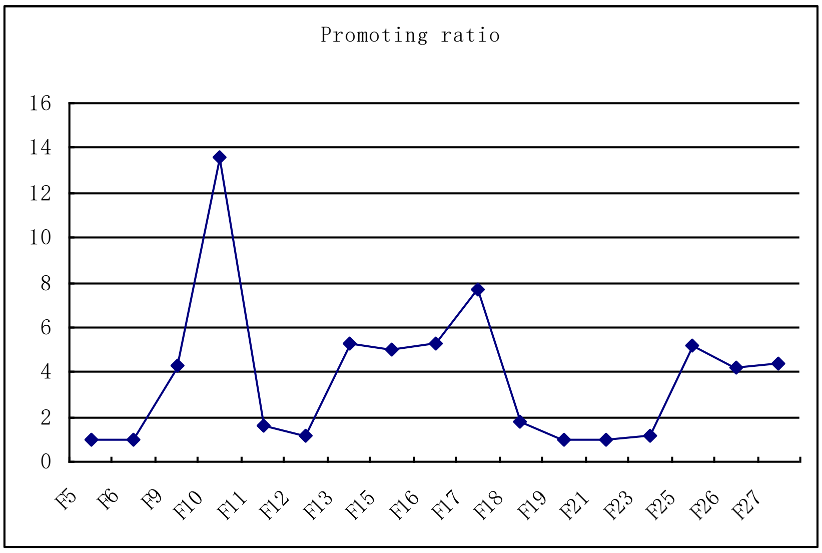

Fig. 2. Promoting ratio of parallel computing vs. sequential computing.

4 Conclusion and Expectation

With the development of multi-core computer, parallel computing is going to be popular and widely applied. Parallelization of sequential algorithms is going into the daily work of algorithm designers and programmers. Through the parallelization of the sequential algorithm, we demonstrate the kernel work of parallelization: to subdivide the computing task into subtasks. Maple experiments show that, our work did increase the computational efficiency. We hope the work can be a reference to the algorithm designers and more valuable algorithms can be shown.

Conflict of Interest

The author declares that there is no conflict of interests regarding the publication of this article.

Author Contributions

Professor Jianhui LI contributes the most research work of the paper.

Master Manlan LIU contributes the data processing and algorithm parameter adjustment work of the paper.

Acknowledgment

The research is supported by special projects in key areas of ordinary universities in Guangdong Province (new generation of information technology) under no. 2020ZDZX3104

References

[1] Carl, P. (1996) A tale of two sieves. *Notices of the AMS*, *43(12)*, 1473-1485.

[2] Aoki K. (2010) Advances in integer factoring technique: The way to factor RSA-768. *Ipsj Magazine*, *51*,1030-1038.

[3] Brent, R P. (1990) Vector and parallel algorithms for integer factorization. *Proceeding of The Third Australian Supercomputer Conference* (pp. 1-22). Australian: Sydney.

[4] Wolski, E., Filho, J. G. S., & Dantas, M. A. R. (2001). Parallel implementation of elliptic curve method for integer factorization using message-passing interface. Retrieved from: http://www.lbd.dcc.ufmg.br/colecoes/sbac-pad/ 2001/007.pdf.

[5] Mcmath, S. S. (2005) Parallel integer factorization using quadratic forms. U.S.N.A-trident Scholar Project Report (Report No.339). Annapolis: US Naval Academy.

[6] Mcmath, S., Crabbe, F., & Joyner, D. (2006) Continued fractions and parallel SQUFOF. *Mathematics*, *1*, 19-38.

[7] Sameh, D., & Ibrahim, G. (2014). A parallel line sieve for the GNFS algorithm. *International Journal of Advanced Computer Science and Applications*, *5(7)*, 178-185.

[8] Anuja, A. S. (2002) Asymmetric key distribution. Graduate Faculty of North Carolina State University.

[9] Wilfrid, K. (2020). Prime factors k2n + 1 of Fermat numbers Fm and complete factoring status, Retrieved from: http://www. prothsearch.com /fermat.html.

[10] Wilfrid, K. (2020). Distributed search for fermat number divisors. Retrieved from: http://www.fermatsearch.org/

[11] Wang, X. (2020). Algorithm available for factoring big fermat numbers. *Journal of Software*, *15(3)*, 86-97.

[12] Li, J. H. (2018). A parallel probabilistic approach to factorize a semi-prime. *American Journal of Computational Mathematics*, *8*, 175-183.

[13] Wang, X. (2017) Strategy for algorithm design in factoring RSA numbers. *IOSR Journal of Computer Engineering*, *19(3)*, 1-7.

Jianhui Li was born in Hunan, China. He received his Ph.D. from Hunan Agriculture University. Since 2010, he had been a staff in charge of researching and developing network security technologies in the Neusoft Institute of Guangdong. Now he has been a professor in Foshan Polytechnic. Professor Li holds the title of high-level talent awarded by Foshan City and is a scientific research partner of China Sunway TaihuLight Supercomputing Center. Up to now, he has presided over more than 10 scientific research projects, published more than 20 papers related to information security and network, and obtained 20 authorized patents. His research interests are information security and Block chain development.

Manlan Liu was born in Hunan, China. She received her mater from Nanchang Hangkong University. Since 2018, she had been a staff in charge of bigdata management and application in the Neusoft institute of Guangdong. Now she has been a lecturer in Foshan Polytechnic. Her research interests are information security and big data management and application.