A parallel line sieve for the GNFS Algorithm

Sameh Daoud
Computer Science Division,
Department of Mathematics
Faculty of Science, Ain Shams University, Cairo, Egypt
Email: Sameh_Daoud2003@yahoo.com

Ibrahim Gad
Computer Science Division,
Department of Mathematics
Faculty of Science, Tanta University, Tanta, Egypt
Email: gad_12006@yahoo.com

***Abstract*—RSA is one of the most important public key cryptosystems for information security. The security of RSA depends on Integer factorization problem, it relies on the difficulty of factoring large integers. Much research has gone into problem of factoring a large number. Due to advances in factoring algorithms and advances in computing hardware the size of the number that can be factorized increases exponentially year by year. The General Number Field Sieve algorithm (GNFS) is currently the best known method for factoring large numbers over than 110 digits. In this paper, a parallel GNFS implementation on a BA-cluster is presented. This study begins with a discussion of the serial algorithm in general and covers the five steps of the algorithm. Moreover, this approach discusses the parallel algorithm for the sieving step. The experimental results have shown that the algorithm has achieved a good speedup and can be used for factoring a large integers.**

Keywords*—parallel Algorithm; Public Key Cryptosystem; GNFS Algorithm.

I. Introduction

Factoring is very important in the field of cryptography, specifically in the RSA cryptosystem. The RSA algorithm [5] is the most popular algorithm in public-key cryptosystems and RSA is used in real world applications such as: internet explorer, email systems, and online banking [12]. The security of RSA algorithm relies on the difficulty of factoring large integers. There are many integer factorization algorithms used to factor large numbers, such as Trial division [6], Pollards p-1 algorithm [7], Lenstra Elliptic Curve Factorization (ECM) [8], Quadratic Sieve (QS) [9] and General Number Field Sieve (GNFS) algorithm [1]–[4]. GNFS is the best known algorithm for factoring large composite numbers over than 110 digits. This algorithm takes a long time to factor large integers. Therefore, this paper presents an implementation of parallel GNFS algorithm on a BA-cluster.

The main objective of this paper giving new proposed algorithm for sieving step in cluster system. This paper consists of eight sections. Section II will introduce the GNFS algorithm. Section III gives the reasons for selecting sieve step. Section IV gives an overview for serial sieve step and give an overview for previous parallel sieve step. Section V proposes a new method for parallel sieve step on cluster system. In section VI the configuration of hardware and software used to implement the parallel sieving step on cluster system. Section VII introduces the experimental results for the proposed methods. Section VIII focus in conclusion and future works.

II. The GNFS Algorithm

The General Number Field Sieve (GNFS) algorithm [1], [2] is derived from the Number Fields Sieve (NFS) algorithm, developed by A. K. Lenstra, H. W. Lenstra, M.S. Manasse and J. M. Pollard [10].

GNFS have five major steps which are described as follows:

1) **Step 1**: (***Polynomial selection***)
Find a polynomial $f : \mathbb{R} \to \mathbb{R}$ of degree d with integer coefficients as follows:

$$f(x) = a_d x^d + a_{d-1} x^{d-1} + \cdots + a_0 \qquad (1)$$

such that $f(m) \equiv 0(\mod n)$.

2) **Step 2**: (***Factor bases***)
The main objective of this step is to find three types of factor bases: (1) rational factor base, R, (2) algebraic factor base, A, and (3) quadratic character base, Q. The three factor bases are define as follows:
The rational factor base R [1]
A rational factor base is a finite collection of prime numbers, p_i, up to some bound M, $M \in \mathbb{N}$. i.e.

$$R = \{p : p \text{ is a prime and } p \leq M,\ M \in \mathbb{N}\ \}.$$

Smooth over R [1]
An integer $l \in \mathbb{Z}$ is said to be smooth over a rational factor base R if R contains all of the prime divisors of l. i.e.

$$l = \prod_{p_i \in R} p_i.$$

The algebraic factor base A [1]
An algebraic factor base is a finite set $\{a+b\theta\} \subset \mathbb{Z}[\theta]$ where for $a, b \in \mathbb{Z}$, each $a+b\theta$ satisfies $\forall (a, b), \nexists$ $c, d \in \mathbb{Z}[\theta]$ such that $c \cdot d = a + b\theta$.
Smooth over A [1]
An element $l \in \mathbb{Z}[\theta]$ is said to be smooth over an algebraic factor base A if

$$l = \prod_{(c,d) \in \grave{A} \subset A} (c + d\,\theta).$$

The quadratic character base Q [1]
The quadratic character base Q is a finite set of pairs (p, r) with the same properties as the elements of the algebraic factor base, but the primes $p_i \in Q$ are larger than the largest in the algebraic factor base, $p_i > \grave{p} \in A$ where $\grave{p}$ is the largest element in the algebraic factor base A, i.e.

Digits	Sieve	Total	Sieve/Total
30	26.5s	32.3s	82%
39	15.0s	19.7s	76.1%
45	184.3s	250s	74%
51	222.3s	311.5s	71.4%
61	3620.7s	4320.4s	84%
76	26477.9s	37171.9s	71.2%
98	17300.6s	20790.9s	83.2%

TABLE I: GNFS integer factorization records [14]

$$Q = \{(p_i, r_i) : p_i > \hat{p} \text{ where } \hat{p} \text{ the largest in the algebraic factor base } A\}$$

3) **Step 3**: (***Sieving***)
Find many pairs of integers (a, b) with the following properties:
1. $\gcd(a, b) = 1$.
2. $a+bm$ is smooth over the rational factor base.
3. $a+b\theta$ is smooth over the algebraic factor base.

4) **Step 4**: (***Linear algebra***)
The relations are put into relation sets and a very large sparse matrix is constructed. The matrix is reduced resulting in some dependencies, i.e. elements which lead to a square modulo n.

5) **Step 5:** (***Square root***)

- Calculate the rational square root, r, such that:
$$r^2 = \prod_{(a,b)\in V} (a + bm)$$
- Calculate the algebraic square root, s, such that:
$$s^2 = \prod_{(a,b)\in V} (a + b\theta)$$
- Then p and q can then be found by $\gcd(n, s-r)$ and $\gcd(n, s + r)$ where p and q are the factors of n.

III. Why Sieving Step?

The main objective of this section is to give the importance of the sieving step. Previous studies shows that the sieving step is very important for several reasons:

1) The sieving step is the most time consuming, it takes more than 70% of the total time from the time of implementation as shown in Table I [14].
2) The second reason is that the sieving step can be parallelized easily.

The experimental studies show that there are some problems in the implementation that led to slow the previous parallel program. In the previous algorithm there are many communications between the master nodes and the slaves. The communication times increase when the size of n increases. Another cause for inefficiency is that each processor does sieving for different pairs. Therefor, the sieving time for each processor might be different. The master node can not start the next sieving until all the slave nodes finish their sieving [14].

IV. Previous Sieving Work

Algorithm 1 shows the steps of serial sieving. The sieving step uses nested for-loops, one for the values of $b's$ and the other for the values of $a's$. In the outer loop, b ranges from $-C$ to C, usually the values of $b's$ are in range $1 \leq b < C$. In the inner loop, b is fixed and a changes from $-N$ to N. The sieving step takes long time because it uses two loops and the values of a and b are usually very large.

Algorithm 1 Serial sieving algorithm [14].

```
1: b_0 = 1;
2: b_1 = C;
3: a_1 = -N;
4: a_2 = N ;
5: for (b = b_0; b < b_1; b++) do
6:     for (a = a_1; a < a_2; a++) do
7:         if (a, b) Smooth over R and Smooth over A then
8:             save(a, b);
9:         end if
10:    end for
11: end for
```

L.T.Yang, L.Xu, and M.Lin proposed parallel sieving in a cluster system [12]–[15]. The basic idea of the proposed algorithm is that each processor takes a range of $b's$ values and generate a set of (a, b) pairs as shown in algorithm 2.

Algorithm 2 Parallel sieving algorithm [14].

```
1: MPI_Init();
2: MPI_Comm_size();
3: MPI_Comm_rank();
4: b_0 = Min_b;
5: b_1 = Max_b;
6: a_1 = -N;
7: a_2 = N ;
8: num_of_bs = ((b1 - b0)/p);
9: MPI_Bcast(num_of_bs);
10: for (b=(taskid*num_of_bs +b0)+1;b<=(b0+(taskid+1)*
    num_of_bs;b++) do
11:     for (a = a_1; a <= a_2; a++) do
12:         if (a, b) Smooth over R and Smooth over A then
13:             if master then
14:                 MPI_Recv((a,b));
15:                 save(a, b);
16:             else
17:                 MPI_Send((a,b));
18:             end if
19:         end if
20:     end for
21: end for
22: MPI_Finalize();
```

V. The new methods

The main objective of this section is to describe the new methods for the parallel sieving step of GNFS algorithm. The new methods improve the parallel sieving algorithm by decreasing the communications between the master node and the slaves. In the following sections (V-A, V-B) we explain the new methods and the results of each method.

A. The first method

The main idea of the first method is to divide the range of b between the processors. This is because, each b in the outer loop generates a set of ordered pairs (a, b) independently of the others $b's$. So, each processor takes a range of b values and generates a set of (a, b) pairs and then saves in a local file, see Fig. 1. When all processors finish the computations of finding their sets of ordered pairs (a, b), the master node copy all the files that have the sets of (a, b) pairs from the slaves into one file.

B. The Second Method

The main idea of the second method is the same as the first method, it depends on dividing the range of b between the processors. Except that each processor takes a range of b values and generate a set of (a, b) pairs and then save it in an array of large size ($rels$), see Fig. 2. Then each slave, find the $rels$ of different sets of ordered pairs (a, b) for each b in the range belonging to this slave, then the slave will send the $rels$ to the master, and the master node receives all the sets of $rels$ from the slaves. This process will be repeated until we reach the last b in the range belonging to this slave.

VI. Hardware and Software Programming Environment

The parallel GNFS program is implemented on a Bibliotheca Alexandrina (BA) Supercomputer which is located in Alexandria library, Alexandria, Egypt. The supercomputer is a high performance computing cluster with performance reaching 11.8 TFLOPS. It is composed of 130 computational nodes, 6 management nodes including two batch nodes for job submission (64 Gbyte RAM), inter-process Communication network, and 36-TByte storage. Each node has two Intel Quad core Xeon 2.83 GHz processors (64 bit technology), 8 Gbyte RAM, 80 Gbyte hard disk, and a GigaEthernet network port.

The parallel code is based on the serial code developed by C. Monico in [3]. The program is written in ANSI C and compiled by GNU C compiler (gcc) and run under Linux operating system. We have used MPI library to write the parallel program. MPICH1 [16] is installed for MPI library. Also we installed a free library GMP [17] which is required to compile and to run the program.

VII. Performance Evaluation

A. Test Cases

In order to test our parallel algorithm for speedup and efficiency, we choose different n and different number of processors. In Table II shows all test cases and number of processors which are used.

Test	Digits of n	Number of processors
1	61	1, 2, 4, 8, 10, 12, ,14, 16
2	76	1, 2, 4, 8, 10, 12, ,14, 16
3	80	1, 2, 4, 8, 10, 12, ,14, 16
4	100	1, 2, 4, 8, 10, 12, ,14, 16
5	110	1, 2, 4, 8, 10, 12, ,14, 16
6	120	1, 2, 4, 8, 10, 12, ,14, 16
7	130	1, 2, 4, 8, 10, 12, ,14, 16

TABLE II: Test cases and number of processors

B. Timing Results

The time for the first method and the second method for each test case is shown in Fig.3.

The Fig.3 show that the ruining time decreases by increasing the number of processors. From Fig.3 the first method is faster than the second method using small number of processors, otherwise the two methods are approximately equal.

C. Speed-Up

Speedup is defined by the following formula: $S_p = \frac{T_1}{T_p}$, where p is the number of processors, T_1 is the execution time of the sequential algorithm, and T_p is the execution time of the parallel algorithm with p processors. The speedup for the test cases using different number of processors for the first method and for the second method are presented in Fig.4.

From Fig.4 the second method is better than the first method when n is small number, the first method is better than the second method when n is large number.

D. Sieving Efficiency

Efficiency is defined as $E_p = \frac{S_p}{p} = \frac{T_1}{pT_p}$. It is a value, between zero and one. The sieving efficiency for each test case is shown in Fig.5.

From Fig.5 the first method is better than the second method using small number of processors, otherwise is approximately equal.

VIII. Discussions

We propose two algorithms for sieving step in cluster system. The difference between them is that one generate a set of (a, b) pairs and then save it in local file for each processor, the other strategy is to generate a set of (a, b) pairs and then save it in an array of large size then send the set of (a, b) pairs to the master.

The experimental studies show that the ruining time decreases by increasing the number of processors. From Fig.3 the first method is faster than the second method when using small number of processors, otherwise they are approximately equal. Fig.4 shows that the speed-up for the second method is better than the first method when n is small number, the first method is better than the second method when n is large number. Fig.5 shows that the efficiency for the first method is better than the second method using small number of processors, otherwise the efficiency is approximately equal.

There are still open questions and some research points which can be studied, in future, such as:

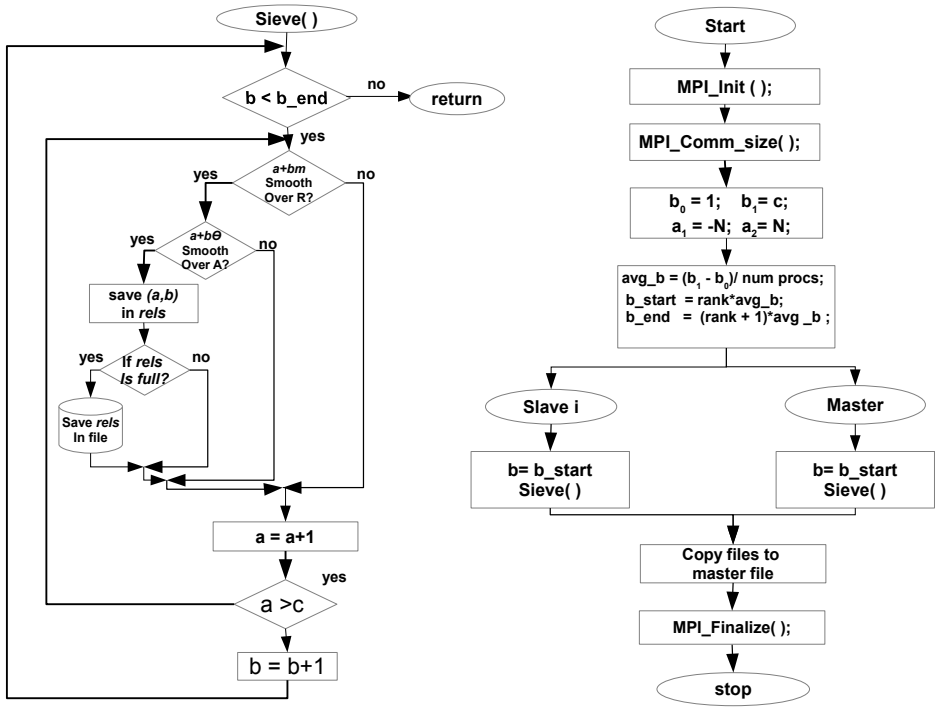

Fig. 1: The flowchart of first method

1) Decreasing the communications time when the size of n increases by decreasing the communications between the master nodes and the slaves, so the sieving time decreases when the communications decreases.
2) Further improvements on better load balance.
3) Trying to make all the steps of the algorithm in parallel whenever possible

REFERENCES

[1] M.Case, *A beginners guide to the general number field sieve*, pages 14, 2003

[2] M.E.Briggs. *An introduction to the general number field sieve*. Masters thesis,Virginia Polytechnic Institute and State University, 1998.

[3] C.Monico, *General number field sieve documentation.*, GGNFS Documentation,Nov 2004

[4] J.Dreibellbis., *Implementing the general number field sieve.*,pages 5–14, June 2003

[5] R.L.Rivest, A.Shamir, and L.M.Adelman,*A method for obtaining digital signatures and public-key cryptosystems.*,Technical Report MIT/LCS/TM-82, 1977.

[6] M.C.Wunderlich and J.L.Selfridge. *A design for a number theory package with an optimized trial division routine*. Communications of ACM, 17(5):272–276, 1974.

[7] J.M.Pollard. *Theorems on factorization and primality testing*. In Proceedings of the Cambridge Philosophical Society, pages 521–528, 1974.

[8] H.W.Lenstra. *Factoring integers with elliptic curves*. Annals of Mathematics(2),126:649–673, 1987.

[9] C.Pomerance. *The quadratic sieve factoring algorithm*. In Proceeding of the EUROCRYPT 84 Workshop on Advances in Cryptology: Theory and Applications of Cryptographic Techniques, pages 169–182. Springer-Verlag, 1985.

[10] H.W.Lenstra, C.Pomerance, and J.P.Buhler. *Factoring integers with the number field sieve*. In The Development of the Number Field Sieve, volume 1554, pages 50–94, New York, 1993. Lecture Notes in Mathematics, Springer-Verlag.

[11] A.K.Lenstra. *Integer factoring*. Designs, Codes and Cryptography, 19(2-3):101–128, 2000.

[12] L.T.Yang, L.Xu, M.Lin, J.Quinn. *A Parallel GNFS Algorithm with the Biorthogonal Block Lanczos Method for Integer Factorization*,Lecture Notes in Computer Science Volume 921, pp 106–120, 2006

[13] L.Xu, L.T.Yang, and M.Lin. *Parallel general number field sieve method for integer factorization*. In Proceedings of the 2005 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA-05), pages 1017–1023, Las Vegas, USA, June 2005.

[14] L.T.Yang, L.Xu, and M.Lin. *Integer factorization by a parallel gnfs algorithm for public key cryptosystem*. In Proceddings of the 2005 International Conference on Embedded Software and Systems (ICESS-05), pages 683–695, Xian, China,December 2005.

[15] L.T.Yang, L.Xu, and S.Yeo and S.Hussain. *An integrated parallel {GNFS} algorithm for integer factorization based on Linbox Montgomery block Lanczos method over GF(2)*. ScienceDirect, Computers & Mathematics with Applications, volume 60, number 2, pages 338 – 346, 2010.

[16] MPICH1: http://www.mpich.org/.

[17] T.Granlund. The GNU Multiple Precision Arithmetic Library (GNU MP). TMG Datakonsult, Boston, MA, USA, 2.0.2 edition, June 1996,http://www.ontko.com/pub/rayo/gmp/gmp_toc.html.

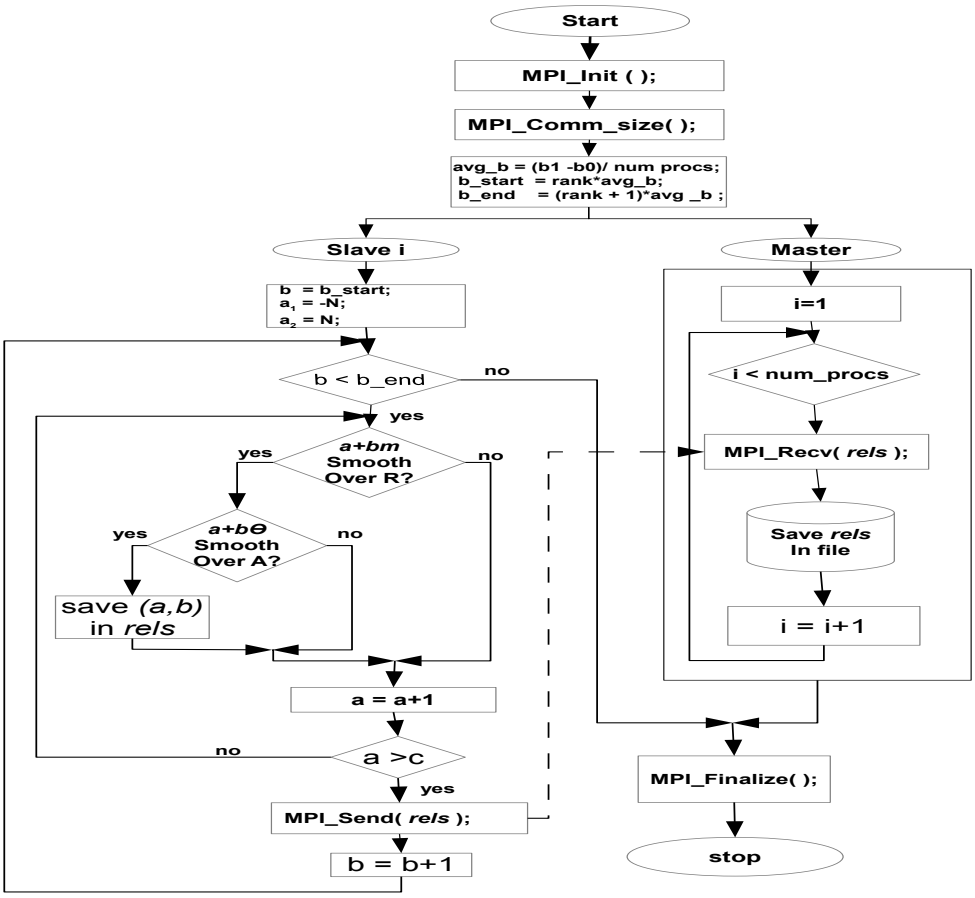

Fig. 2: The flowchart of second method

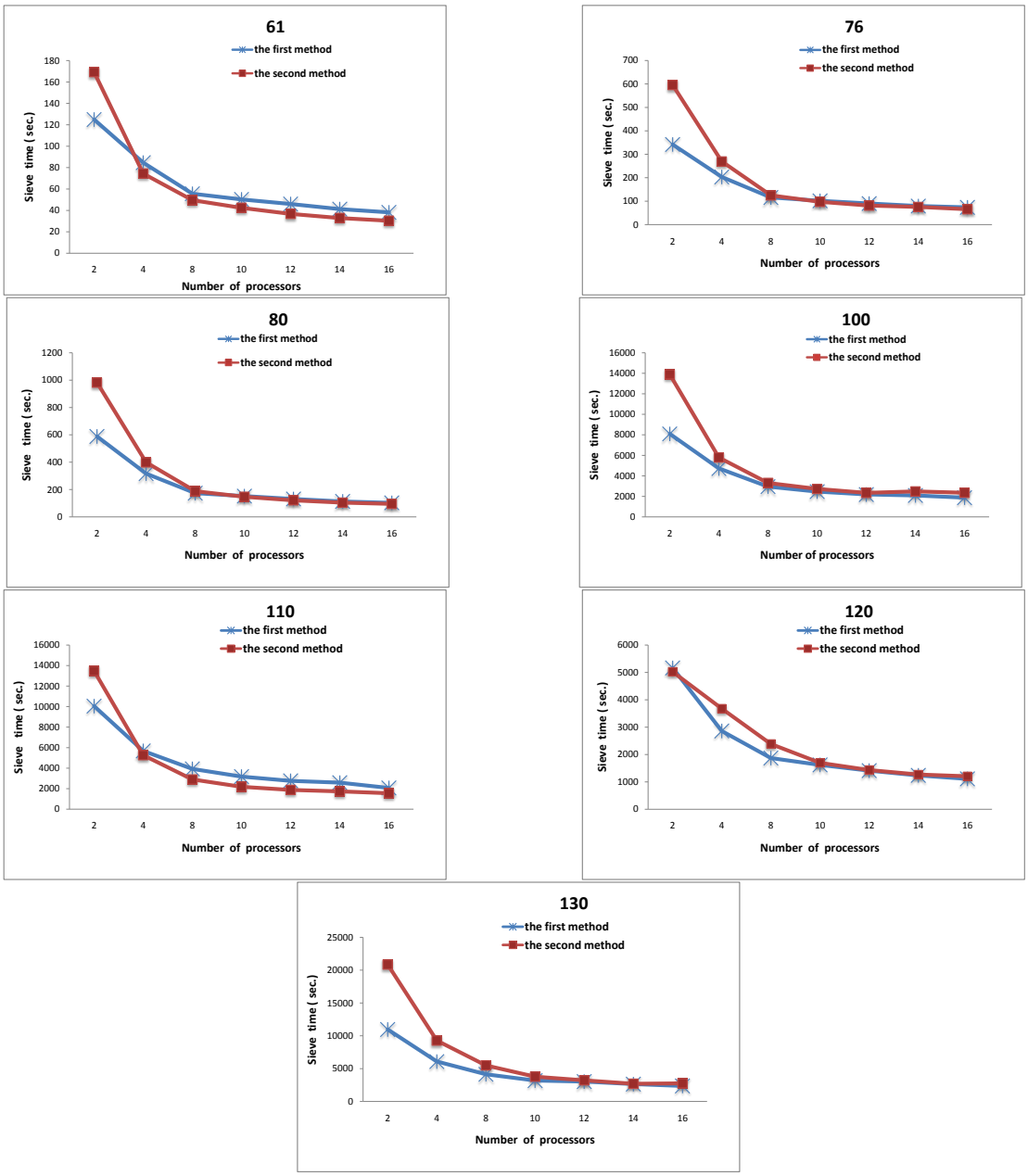

Fig. 3: The parallel implementation for the first and the second method. n = 61, 76, 80, 100, 110, 120, 130 digits

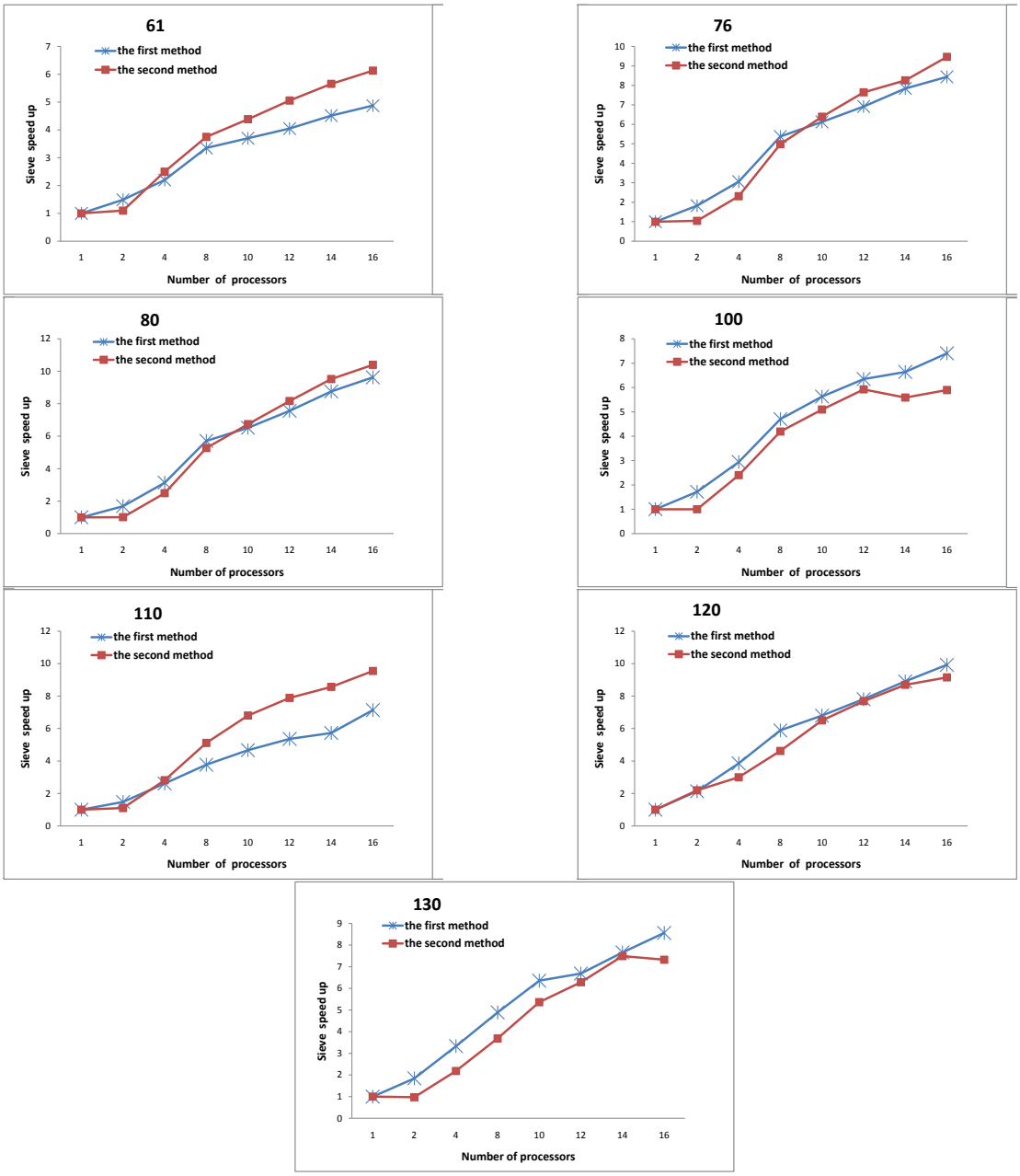

Fig. 4: Sieve Speed-up for the first method and the second method. n = 61, 76, 80, 100, 110, 120, 130 digits

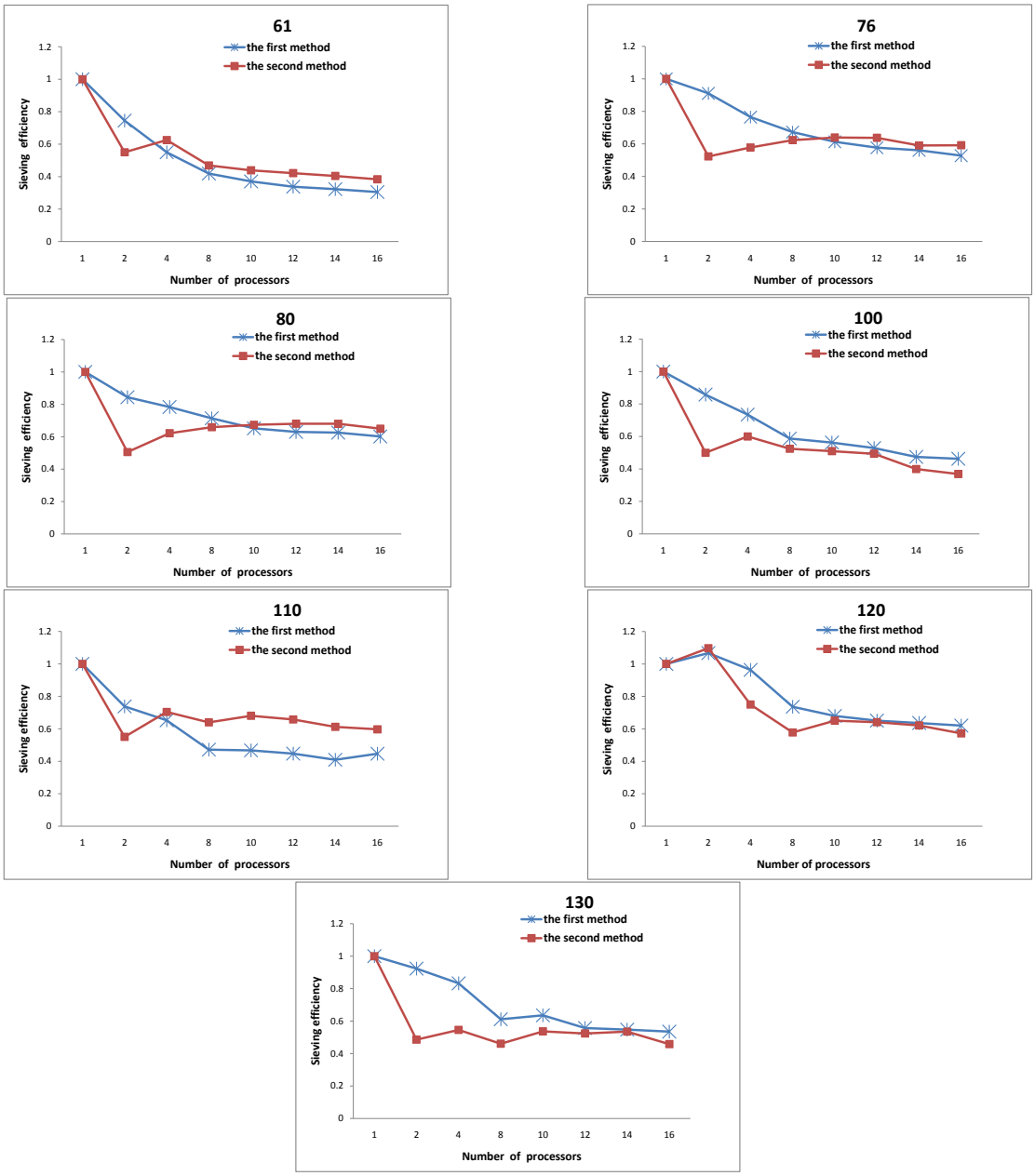

Fig. 5: Sieve efficiency for the first method and the second method. n = 61, 76, 80, 100, 110, 120, 130 digits