Parallelization of Prime Number Generation Using Message Passing Interface

IZZATDIN AZIZ, NAZLEENI HARON, LOW TAN JUNG, WAN RAHAYA WAN DAGANG
Department of Computer and Information Sciences
Universiti Teknologi Petronas
31750 Tronoh, Perak
MALAYSIA

Abstract:-. In this research, we proposed a parallel processing algorithm that runs on cluster architecture suitable for prime number generation. The proposed approach is meant to decrease computational cost and accelerate the prime number generation process. Several experimental results are shown to demonstrate the viability of our work.

Key-Words: - Prime number generation, parallel processing, cluster architecture, MPI, cryptography

1 Introduction

Prime numbers has stimulated much of interest in mathematical field or in security field due to the prevalence of RSA encryption schemes. Cryptography often uses large prime numbers to produce cryptographic keys which are used to encipher and decipher data. It has been identified that a computationally large prime number is likely to be a cryptographically strong prime. However, as the length of the cryptographic key values increases, this will result in the increased amount of computer processing power required to create a new cryptographic key pair. In particular, the performance issue is related to time and processing power required for prime number generation.

Prime number generation comprises of processing steps in searching for and verifying large prime numbers for use in cryptographic keys. This is actually a pertinent problem in public key cryptography scheme, since increasing the length of key to enhance the security level would results in a decrease in performance of a prime number generation system.

Another trade off resulting from using large prime numbers is pertaining to the primality test. Primality test is the intrinsic part of prime number generation and yet the most computational intensive sub process. It has also been proven that testing the primality of large candidates is very computationally intensive.

Apart from that, the advent of parallel computing or processing has invited many interests to apply parallel algorithms in a number of areas. This is because it has been proven that using parallel processing can substantially increase the processing speed. In this paper, we present a parallel processing approach in cluster architecture for prime number generation that would provide improved performance in generating cryptographic keys.

2 Related Work

Despite the importance of prime number generation for cryptographic schemes, it is still scarcely investigated and real life implementations are of rather poor performance [1]. However, a few approaches do exist in order to efficiently generate prime numbers [1-5]. Maurer proposed an algorithm to generate provable prime numbers that fulfill security constraints without increasing the expecting running time [2]. An improvement has been made to Maurer's algorithm by Brandt et al to further speed up the prime number generation [3]. Apart from that, the proposed work has also included a few ways for further savings in prime number generation [3]. Joye et al has presented an efficient prime number generation scheme that allows fast implementation on cryptographic smart card [1]. Besides that, Cheung et al has originated a scalable architecture to further speed up the prime number validation process at reduced hardware cost [4]. All of these researches however, were focusing on processing the algorithm sequentially. It has been proven that tasks accomplished through parallel computation results in faster execution as compared to a computational processes that runs sequentially [9]. Tan et al has designed a parallel pseudo-random generator using Message Passing Interface (MPI)

[5]. This work is almost similar to ours but with different emphasis. The prime numbers generated are to be used for Monte Carlo simulations and not cryptography. Furthermore, considerable progresses have been made in order to develop high-performance asymmetric key cryptography schemes using approaches such as the use of high-end computing hardware [6, 7, and 8].

3 System Model

3.1 Experimental Setup

The experimental cluster platform for performance comprised of 20 SGI machines. Each of the machines consists of off-the-shelf Intel i386 based dual P3-733MHz processors with 512MB memory Silicon Graphics 330 Visual Workstations. These machines are connected to a Fast Ethernet 100Mbps switch. The head node performs as master node with multiple network interfaces [10]. Although these machines are considered to be superseded in terms of hardware and performance as compared to the latest version of high performance computers, what's important in this research is the parallelization of the algorithm and how jobs are disseminated among processors.

3.2 Number Generation

In order to first generate the number, a random seed is picked and input into the program. The choice of seed is crucial to the success of this generator as it has to be as random as possible. Otherwise anyone who uses the same random function would be capable of generating the primes, thus beats the purpose of having strong primes.

3.3 Primality Test

We have selected trial division algorithm as the core for primality testing. This algorithm is based on a given composite integer n, trial division consists of trial-dividing n by every prime number less than or equal to $\sqrt{n}$. If a number is found which divides evenly into n, that number is a factor of n.

3.4 Parallel Approach

Once a random number have been generated, master node will create a table of dynamic 2D array, which later will be populated with odd numbers. As shown in Fig.1, a pointer-to-pointer variable ***table* in master, will points to an array of pointers that subsequently points to a number of rows. This will result in a table of dynamic 2D array. After the table of dynamic 2D array is created, master will then initialize the first row of the table only.

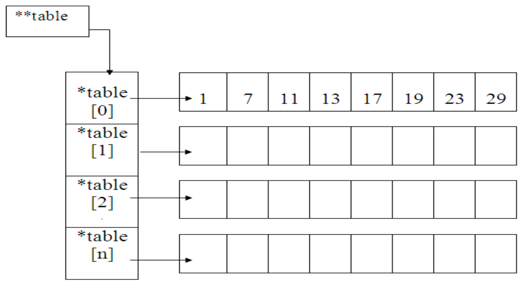

Fig.1 Master creates a dynamic 2D array to be populated with odd numbers

The parallel segment begins when master node broadcasts the row[0] to all nodes by using MPI_Bcast. This row[0] will be used by each node to continue populating the rest of the rows of the table with odd numbers. Master node will then equally divide n-1 number of rows left that is yet to be populated by number of nodes available in the grid cluster. Each node will be given an equal number of rows to be populated with odd numbers. This could be achieved by using MPI_Send. A visual representation of this idea is depicted in Fig.2.

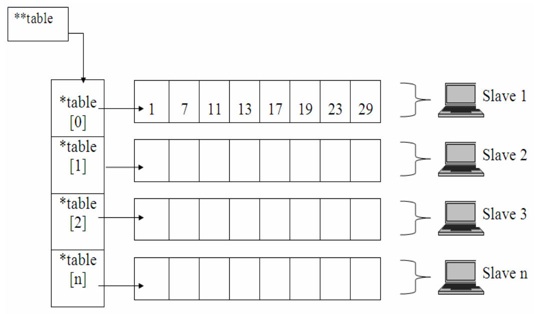

Fig.2. Master sends an equal size of row to each slave

Each node will receive n numbers of rows to be populated with odd numbers. This is where the parallel process takes place. Each node will process each row given concurrently. Each node will first populate the rows with odd numbers. Then they will filter out for prime numbers using the primality test chosen. The odd prime numbers will remain in the rows but those that are not will be assigned to NULL. Each populated row are then returned to master node, whom then randomly pick for three distinct primes for the value of $p,q,$ and public key e of the cryptographic scheme.

For an example, if there are 4 processors available to execute the above tasks, and there are 1200 rows need to be populated with prime numbers, each slave will be given 300 rows to be processed. The overall procedure is depicted in Fig.3.

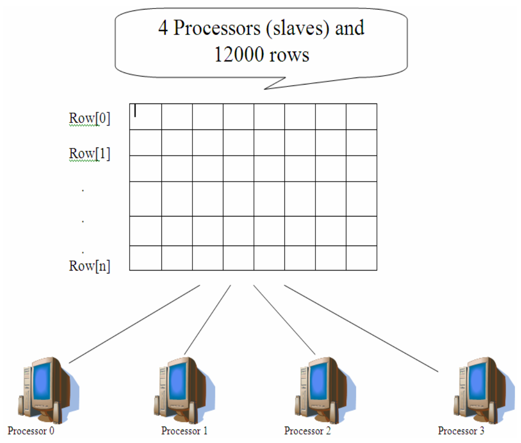

Fig.3.Example of assigning 1200 rows to 4 processors (slaves)

Processor 0 will process row(1) up to row(299), processor 1 will be processing row(300) up to row(599), processor 2 will be processing row(600) up to row(899) and lastly processor 3 will be processing row(900) up to the last row, row(1199).

After each node returns the populated rows to master node, it will then pick randomly prime numbers to be assigned as the value of *p, q,* and *e*. These values can later be used for encryption and decryption part of a cryptosystem algorithm. It is to be reminded that the parallel process that takes place in the whole program is only on the prime number generation.

4 Parallel Algorithm

The algorithm of the parallel program is as follows:

```
Start
Master creates a table of odd numbers and
initialized row [0] only
Master broadcasts row [0] to all slaves
Master sends a number of rows to each slave
    Each slave will receive an initialized
    row from master
    Each slave will populate row prime
    numbers
    Each slave will return populated row to
    Master
Master waits for results from slaves
Master receives populated rows from
each slave.
Master checks unpopulated rows
  If maxRow > 0
      Master sends unpopulated row
      to slave
Master picks prime numbers randomly
Prompt to select program option
Switch (method)
   Case 1: prompt to enter a
   value greater than 10000
      If value > 10000, generate key
      primes
      Else, Exit program
   Case 3: open file and decrypt
   Case 4: exit program
      End
End
```

5 Evaluation

Table 1, shows the execution time of running the parallel program on single and more computing nodes. From the results, it can be inferred that running the algorithm in parallel mode has accelerated the prime number generation process. However, it seems like there is a noticeable increase in processing time when the program is running more than 3 nodes. The execution time has recorded to be higher when more nodes participated in the generation process. This may be caused by the network latency during the distribution of the tasks, which leads to the increased of execution time taken for the communication between nodes.

Table 1: Comparison of Execution Time for Different Number of Nodes.

Number of nodes	Execution Time (ms)
1	7.850
3	0.039
5	0.043
10	0.053
30	0.093

Fig.4 shows the performance measurement using MPI_GATHER tested on 15 nodes. This figure was captured using the MPICH Jumpshot4 tool to evaluate the algorithm usage of MPI libraries. The numbers plotted shows the amount of time taken for each node to send back the prime numbers discovered back to the master node.

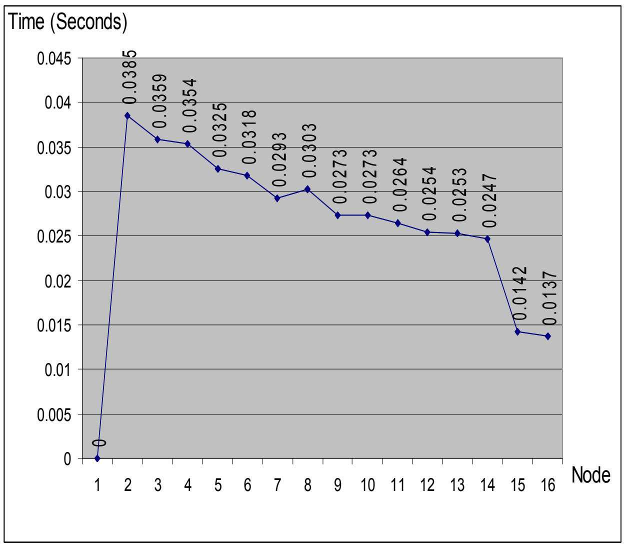

Fig.4. Time taken for MPI_BCAST and MPI_GATHER running on 15 nodes.

From the figure, it is observed that the algorithm gather was massive for the first node and deteriorated as it approached the last node. This may due to the frequent prime numbers discovered at the beginning of the number series and becomes scarce as the numbers becomes larger towards the end. This will prove that the relative frequency of occurrence of prime numbers decreases with size of the number which result in lesser prime numbers were sent back to master node by later nodes.

6 Conclusion

We have proposed a parallel approach for prime number generation in order to accelerate the process. The parallelism and the cluster architecture of this approach have been tested with large prime numbers. Results have demonstrated that improvement can be obtained if parallel approach is deployed. However, further improvements can be made that include:

(1) Use other primality test that is more significant or feasible for large prime number generation such as Rabin-Miller algorithm.

(2) Use other random number generation that can produce random numbers with less computation yet provides higher security level.

References:

[1] M. Joye, P. Paillier and S. Vaudenay, Efficient Generation of Prime Numbers, *Cryptographic Hardware and Embedded Systems*, vol. 1965 of *Lecture Notes in Computer Science,* pp. 340-354, Springer-Verlag, 2000.

[2] Maurer, Fast Generation of Prime Numbers and Secure Public-Key Cryptographic Parameters, *Journal of Cryptology*, vol.8 no.3 (1995), 123-156.

[3] J.Brandt, I. Damgard, and P. Landrock. Speeding up prime number generation. *In Advances in Cryptology -- ASIACRYPT '91*, vol. 739 of *Lecture Notes in Computer Science*, pp. 440--449, Springer-Verlag, 1991.

[4] Cheung, R.C.C., Brown, A., Luk, W., Cheung, P.Y.K., A Scalable Hardware Architecture for Prime Number Validation, *IEEE International Conference on Field-Programmable Technology, 2004.* pp. 177-184, 6-8 Dec. 2004.

[5] Tan, C. J. and Blais, J. A. PLFG: A Highly Scalable Parallel Pseudo-random Number Generator for Monte Carlo Simulations. *8th international Conference on High-Performance Computing and Networking* (May 08 - 10, 2000). *Lecture Notes In Computer Science*, vol. 1823. Springer-Verlag, London, 127-135.

[6] Agus Setiawan, David Adiutama, Julius Liman, Akshay Luther and Rajkumar Buyya, *GridCrypt :* High Performance Symmetric Key Cryptography using Enteprise Grids. *5th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 200)* , Singapore. Springer Verlag Publications (LNCS Series), Berlin, Germany. December 8-10, 2004.

[7] Praveen Dongara, T. N. Vijaykumar, Accelerating Private-key cryptography via Multithreading on Symmetric Multiprocessors. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, March 2003.

[8] Jerome Burke, John McDonald, Todd Austin, Architectural Support for Fast Symmetric-Key Cryptography. *Proc. ACM Ninth Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX),* Nov. 2000.

[9] Selim G Aki, Stefan D Bruda, Improving A Solution's Quality Through Parallel Processing. *The Journal of Supercomputing* archive.Volume 19 , Issue 2 (June 2001).

[10] Dani Adhipta, Izzatdin Bin Abdul Aziz, Low Tan Jung, Nazleeni Binti Haron .Performance Evaluation on Hybrid Cluster: The Integration of Beowulf and Single System Image*, The 2nd Information and Communication Technology Seminar (ICTS),Jakarta.* August 2006.