PARALLELIZING SHAMIR'S SECRET SHARING ALGORITHM*

Joseph K. Arbogast, Isaac B. Sumner, and Michael O. Lam
Department of Computer Science
James Madison University
Harrisonburg, VA 22807
540-568-3335
arbogajk@dukes.jmu.edu, sumnerib@dukes.jmu.edu, lam2mo@jmu.edu

ABSTRACT

This paper describes how Shamir's secret sharing algorithm can be parallelized, decreasing the time required to generate key shares for secrets shared among a large group of participants. Using an open-source C implementation of Shamir's algorithm and the OpenMP multiprocessing programming interface, we parallelized regions of the algorithm and reduced execution time significantly. We were able to see near-linear speedup in both the key share generation phase and the re-combining phase. We also observed weak scaling when generating the key shares. Our work enables more efficient secret sharing using Shamir's algorithm.

INTRODUCTION

Shamir's secret sharing scheme [1] is a method for dividing a secret among a group where a certain threshold of the keys must be combined in order to reproduce the secret. With large secrets, such as an entire text document, this process can be slow and expensive, especially if the number of participants and the threshold are high. Motivation for this work comes from an interest in using parallel systems to improve security. Shamir's secret sharing is useful in scenarios where no single person should have an entire secret. Real world applications are currently seen in the medical field and patient privacy. Medical reports and images are commonly shared using secret sharing schemes [5].

* Copyright © 2017 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

In this paper, we explore opportunities for parallelism in Shamir's algorithm, with a goal of reducing the amount of time taken both to generate and to join shares in a scalable manner.

Scaling in high-performance program analysis is broken into two categories: strong and weak scaling. Strong scaling describes how performance changes as the number of processes or threads increases for a fixed problem size, while weak scaling examines how performance changes as both the processor/thread count and problem size increase.

Background

Shamir's secret sharing algorithm [1] requires two parameters: the number of shares desired (n) and the threshold that is required to unlock the secret (t). The algorithm computes the shares by generating a random polynomial equation of degree t-1. The secret becomes the constant value in the polynomial equation.

Formula 1: $4x^3 + 9x^2 + 3x + secret$

The required threshold to reproduce the secret of the function listed in Formula 1 would be 4 shares, because t = 4 yields a degree 3 (t-1) polynomial. After generating the random polynomial, the algorithm computes X and Y coordinate pairs by generating a random X value and plugging it into the polynomial function to get a corresponding Y value. In the implementation we used, an extra step is taken and the entire polynomial is modulo with the prime 257. This solves the problem of an attacker gaining information about the secret with each key that they find, by using finite field arithmetic in lieu of integer arithmetic. Each character of the input data is run through this equation, which is repeated n times. The XY pairs become the shares that are distributed to each individual in the group. Finally, Lagrange interpolating polynomials are used to join the shares together and reproduce the original secret.

The problem with this approach (when computed serially) is that a XY ordered pair for each character of the input data must be computed n number of times. This process is slow and provides opportunities for data parallelism.

Project goals

Using an open-source C implementation of Shamir's secret sharing algorithm [2], we explored the benefits of parallelizing the algorithm. The overall focus of our project was to speed up the process of generating shares for large files between a large number of parties. We do not vouch for the security of the implementation we used, only that we were able to speed it up with concurrency.

METHODS

We used OpenMP in order to take advantage of its parallel for loop construct. Specifically, we identified 3 regions of the code where parallelism could be exposed and

implemented parallel versions of these functions.[1] All of our experiments were run on a Dell server with an 8-core (2.4Ghz w/ hyperthreading) Xeon E5-2630v3 processor with 32GB RAM using the maximum number of shares and threshold the original program was capable of generating, which was 255 key shares. For our test data sets, we used text files containing 540, 1080, 2160, 4320, and 8640 characters. We also used a 4096 bit RSA private key, containing 3,272 characters including the RSA header details as an input file into the program. Our weak scaling tests consisted of doubling the character count of the input file while simultaneously doubling the thread count.

Our focus at first was studying the functions that dealt with generating the key shares. We identified two functions in the implementation that allowed for substantial decreases in time for computing the shares. Firstly, we were able to parallelize the for loop that generates the random coefficients used in the polynomial function. Secondly, we were able to parallelize the for loop that handles computing the key shares. By leaving the join_shares function untouched, we were able to verify the correctness of this parallelization, because the join_shares function could reassemble the shares into the original text file. After implementing parallelism in the share generation stage of Shamir's secret sharing scheme and verifying the correctness of that transformation, we switched our focus to implementing parallelism in the join_shares function.

The challenge with parallelizing the shares joining stage of Shamir's secret sharing is correctly identifying OpenMP variable scope (i.e., selecting the variables that should be visible to all threads) as well as identifying and annotating critical regions to prevent race conditions. The most important region to consider is where each thread updates the secret after computing Lagrange interpolating polynomials. We had to synchronize access to this region using the OpenMP critical construct, enabling us to parallelize the for loop that computes the Lagrange interpolating polynomial of the function used in joining the shares back together.

RESULTS

We were able to significantly speed up the secret splitting and joining using OpenMP. Our approach shows excellent strong and weak scaling. This section details some of our results. The times reported in this section are the minimum across all observations, and we observed very little variation between runs.

Strong Scaling

In Figure 1 and Table 1, we show that the times to create the key shares is nearly halved every time we double the number of threads, which means the new implementation achieves close to linear speedup.

[1]Source code available at: github.com/arbogajk/470_SP

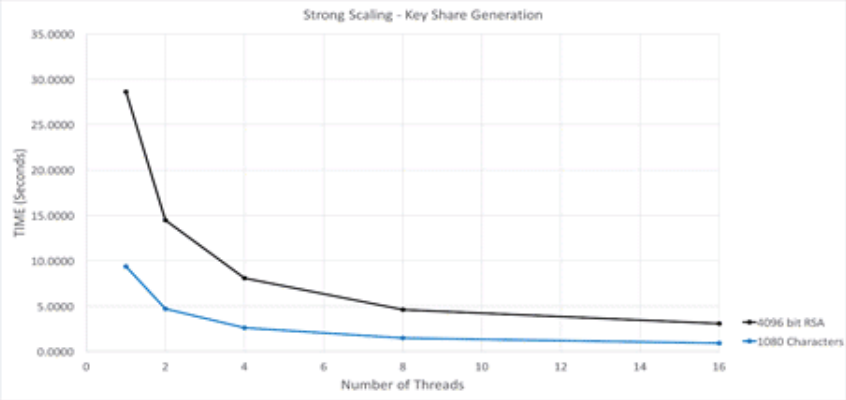

Figure 1: Results of Generating 255 shares with a required unlock threshold of 255 for a 4096 bit RSA key and 1,080-character input file.

Share Generation Results

Threads	4096 Bit RSA Key	1080 Character File
1	28.62 sec	9.36 sec
2	14.46 sec	4.71 sec
4	8.07 sec	2.61 sec
8	4.60 sec	1.46 sec
16	3.09 sec	0.92 sec

Table 1: Shows the times taken to generate 255 shares with a threshold of 255

We get similar scaling results when joining the shares back together to recover the original secret, shown in Figure 2 and Table 2. It's important to note here that we start to see increases in time at 16 threads when joining the shares back together. This corresponds to the number of physical cores our test machine (eight, with sixteen hyperthreads). At this point, increasing the number of threads becomes counterproductive because they cannot all run in parallel on the hardware.

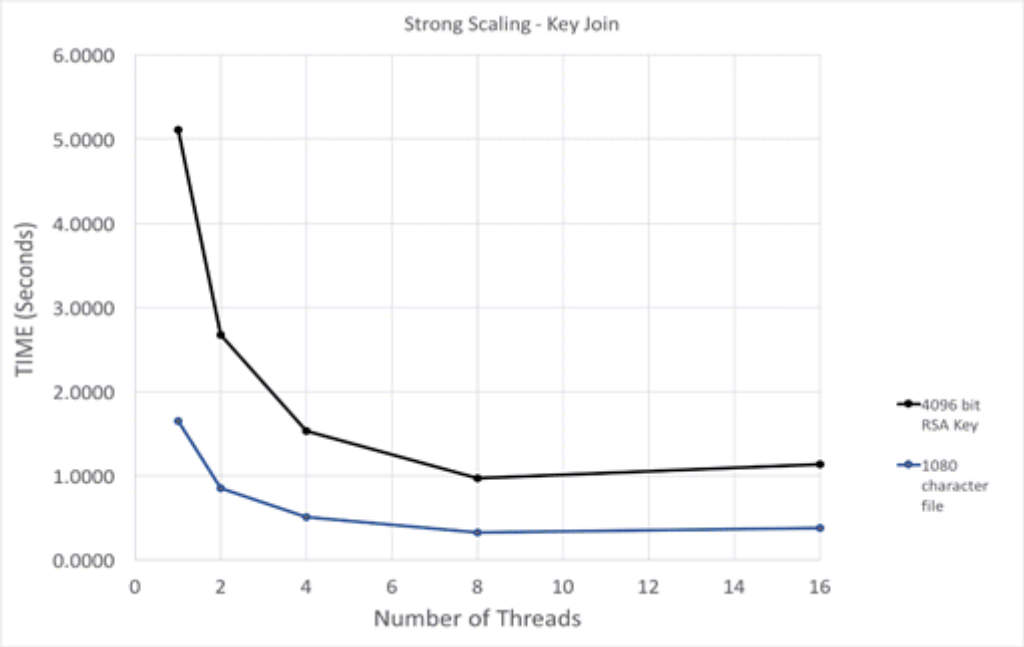

Figure 2: Results of joining all 255 shares to reproduce the secret from the RSA key and the 1,080-character file.

Key Join Results

Threads	4096 Bit RSA Key	1080 Character File
1	5.10 sec	1.64 sec
2	2.67 sec	0.84 sec
4	1.53 sec	0.50 sec
8	0.96 sec	0.32 sec
16	1.13 sec	0.37 sec

Table 2: Times taken to reassemble the 255 shares to reproduce the secret

Weak Scaling

We discovered a second inner loop in the split_string function that provided beneficial results in our weak scaling tests. Table 3 shows that the times do not increase proportionally as we doubled both the input size and the number of threads.

Weak Scaling Test Results

Threads	Character Count	Time
1	540	4.66 sec
2	1080	4.71 sec
4	2160	5.28 sec
8	4320	5.86 sec
16	8640	7.41 sec

Table 3: Times taken after parallelizing the second loop in `split_string` function.

CONCLUSION

We were successful in parallelizing Shamir's secret sharing algorithm, achieving near linear speedup using OpenMP. Future work in parallelization of Shamir's secret sharing could look into distributed memory architectures using message-passing frameworks like MPI, exploring whether the communication between nodes would be a limiting factor on speedup. Rabin's secret sharing algorithm [3] would also be a good candidate for parallelism in future research. This sharing scheme operates under the assumption that each participant can broadcast a message, and that each pair of participants can communicate secretly. We hope our work can serve as a stepping stone for future projects, as there is still a lot that can be done with secret sharing algorithms in the context of parallel and distributed systems.

The source code for this project is available online: github.com/arbogajk/470_SP

References

[1] D. Bogdanov, "Foundations and properties of Shamir's secret sharing scheme," Research Seminar in Cryptography, pp. 1-10, 2007. [Online]. Available: https://pdfs.semanticscholar.org/540b/faa26cfafde5be79aadde37cb79f9d2daf76.pdf

[2] F. T. Penney. (). Original c implementation of Shamir's secret sharing algorithm. Original source code, [Online]. Available: https://github.com/fletcher/c-sss

[3] T. Rabin and M. Ben-Or, "Verifiable secret sharing and multiparty protocols with honest majority," in Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing, ser. STOC '89, Seattle, Washington, USA: ACM, 1989, pp. 73-85,

isbn: 0-89791-307-8. [Online]. Available:
http ://doi.acm.org/10.1145/73007.73014

[4] "OpenMP Application Programming Interface". OpenMP Architecture Review Board, 2015. [Online]. Available:
http://www.openmp.org/wp-content/uploads/openmp-4.5.pdf

[5] R. Basavegowda and S. Seenappa, "Electronic Medical Report Security Using Visual Secret Sharing Scheme," 2013 UKSim 15th International Conference on Computer Modelling and Simulation, April 2013, Cambridge: IEEE. [Online]. Available: http://ieeexplore.ieee.org/document/6527394/