OMEGA NETWORK HASH CONSTRUCTION

By

CHUAH CHAI WEN

Thesis submitted in fulfillment of the
requirements for the degree of
Master of Science

June 2009

TABLE OF CONTENTS

LIST OF TABLES **PAGE**

LIST OF FIGURES **PAGE**

SENI BINA CINCANGAN BERPANDUKAN RANGKAIAN OMEGA

ABSTRAK

Fungsi cincangan kriptografi merupakan primitif kriptografi yang sangat umum dan penting. Ia biasanya digunakan dalam proses pengesahan keaslian dan keutuhan data. Seni binanya adalah berasaskan seni bina Merkle-Damgard yang menerima input panjang berubah dan menghasilkan nilai cincangan panjang tetap. Seni bina Merkle-Damgard yang asas beroperasi secara berjujukan terhadap input data, dan hal ini boleh menimbulkan masalah apabila saiz input yang digunakan adalah besar kerana masa pengiraan juga akan meningkat secara linear. Oleh yang demikian, satu seni bina alternatif yang dapat mengurangkan masa pengiraan amat diperlukan terutama sekali dalam dunia hari ini yang mana pemproses multiteras dan pengaturcaraan multibebenang begitu lumrah. Seni bina Cincangan berpandukan Rangkaian Omega yang mampu beroperasi secara selari pada pemproses multiteras telah dicadangkan sebagai suatu alternatif kepada seni bina Merkle-Damgard. Seni bina Cincangan berpandukan Rangkaian Omega ternyata menunjukkan prestasi yang lebih baik daripada seni bina Merkle-Damgard, dan seni bina pilih aturnya menunjukkan bahawa tahap keselamatannya dari segi penghasilan nilai *digest* sembarangan adalah lebih baik daripada seni bina Merkle-Damgard.

OMEGA NETWORK HASH CONSTRUCTION

ABSTRACT

Cryptographic hash functions are very common and important cryptographic primitives. They are commonly used for data integrity checking and data authentication. Their architecture is based on the Merkle-Damgard construction, which takes in a variable-length input and produces a fixed-length hash value. The basic Merkle-Damgard construction runs over the input sequentially, which can lead to problems when the input size is large since the computation time increases linearly. Therefore, an alternative architecture which can reduce the computation time is needed, especially in today's world where multi-core processors and multithreaded programming are common. An Omega Network Hash Construction that can run parallel on multi-core machine has been proposed as alternative hash function's construction. The Omega Network Hash Construction performs better than the Merkle-Damgard construction, and its permutation architecture shows that its security level in term of producing randomness digest value is better than Merkle-Damgard construction.

LIST OF ABBREVIATION

IEEE	Institute of Electrical and Electronics Engineers
MD	Merkle-Damgard Construction
NIST	National Institute of Standards and Technology
ONHC	Omega Network Hash Construction
RAM	Read Access Memory
SHA	Secure Hash Algorithm
T	Thread

CHAPTER 1

INTRODUCTION

1.1 Introduction to Hash Functions

As the Internet rapidly grows, and bandwidth rapidly increases, cryptography is becoming more and more important for ensuring various types of security over insecure connections. Among data security primitives, data integrity check and data origin authentication are the two most common security services that must be applied in many electronic applications, such as electronic commerce, electronic financial transactions, software distribution, electronic mail, data storage and others. Data integrity check is accomplished through the use of cryptographic hash functions, which operate at the root of many other cryptographic methods in achieving these security services.

The basic operation of a hash function is to transform a variable-size input or message into a fixed-length string called a "hash value" or "message digest." A hash value is generated by a function H of the form $H(M) = n$, where n is the hash value and M is the variable-length input or message. Hash functions are one-way functions; it is easy to generate n from a given M, but given only n, it is computationally infeasible to generate M. Hash functions are designed to produce unambiguous and condense message digests that are uniquely identifiable with their source messages. However, the source messages cannot be deduced from the message digests, and for this reason, the hash function is sometimes known as a digital fingerprint.

Hash functions are built upon the Merkle-Damgard construction [9, 10, 12, 21], which divides an input into equal-size message blocks and passes each block sequentially to a function that processes the message block. The function returns a vector value, which is then passed back to the function for the next message block. A pre-defined vector value is passed with the first message block.

Many researchers are trying to develop better-performing hash functions. The sequential architecture of Merkle-Damgard is recognized as a critical factor for overall hashing performance. However, these researchers are not trying to improve the Merkle-Damgard construction, but are instead trying to improve specific hashing algorithms, such as the Secure Hash Algorithm (SHA).

This research proposes the Omega Network Hash Construction as an alternative hashing architecture to the Merkle-Damgard Construction. Because of the design of the Omega Network Hash Construction, the original inputs cannot be retrieved from their corresponding hash values. In the era of multithreading and multi-core technology, the Omega Network Hash Construction runs in parallel to improve the hashing performance. The goal of this architecture is to improve performance without sacrificing the security provided by the existing Merkle-Damgard architecture.

1.2 Motivation

Hash functions, such as SHA-512, are essential for data integrity assurance and data authentication. For example, they can be used to determine whether any changes have been made to a message or a file. However, since a Merkle-Damgård construction accepts an arbitrary-length input and produces a fixed-length hash value, the computational time to produce a hash value will increase linearly based on the size of the input. Because input sizes can be very large, a linear runtime may be unacceptably slow. This research seeks to enhance the performance of hash construction by using the Omega Network architecture. The SHA-512 hash function is used as the case study for this research because of its popularity.

1.3 Problem Statement

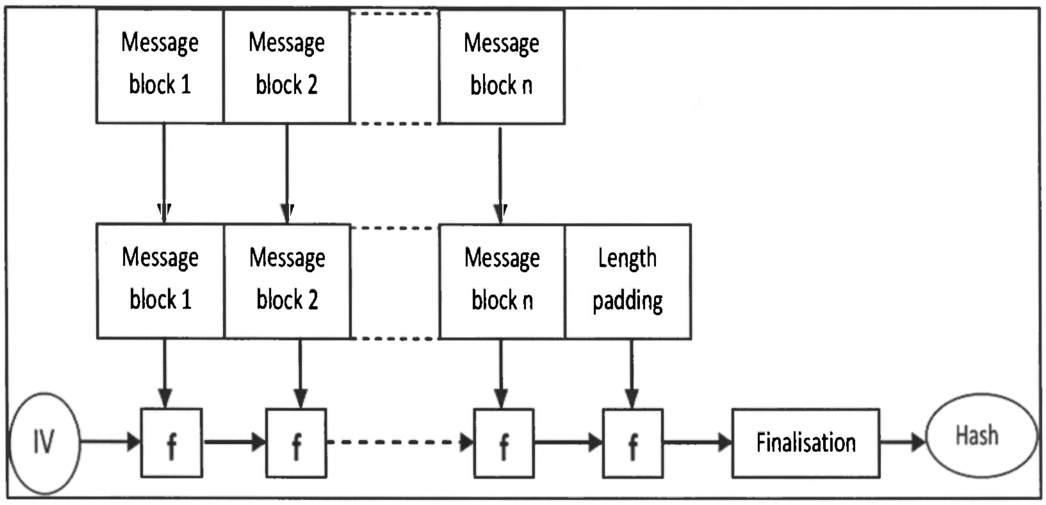


Figure 1.1: Merkle-Damgård Construction [adopted from 9, 10, 12]

The architecture of most hash functions is based on the Merkle-Damgard construction (see Figure 1.1), which is sequential in nature. This means that when the size of the input increases, the computational time will increase linearly. Each step in the Merkle-Damgard construction processes a message block and returns a vector. The first vector is pre-defined, but the remaining vectors are fully dependent on the previous function's output, which slows down the runtime. This in turn has a major effect on the performance of SHA, for example. There is a need to enhance the performance and efficiency of hashing.

1.4 Hypothesis and Research Questions

The proposed Omega Network Hash Construction run parallel in multi-core machine can enhance the speed up and efficiency of SHA while maintaining the security level provided by the existing Merkle-Damgard architecture. The research questions that this thesis will answer are as follows:

- Is the computational time of the Omega Network Hash Construction better? If so, by how much?
- In terms of performance and security, is the proposed architecture as good as or better than the existing Merkle-Damgard construction?
- What are the strengths and weaknesses of the proposed Omega Network Hash Construction?

- Is it worth having an alternative architecture such as the proposed Omega Network Hash Construction?

1.5 Research Objectives

The objectives for this research are as follows:

1. To maintain SHA's security level provided by the existing Merkle-Damgard architecture while using the Omega Network Hash Construction.
2. To use the parallel method construct the Omega Network Hash Constructions to enhance the performance of SHA.
3. To determine which sizes of Omega Network Hash Construction perform well in the multi-core processors' machine.

1.6 Research Scope

This research will focus on enhancing the performance of SHA, specifically SHA-512, in terms of speed up efficiency and overhead reduction. The first column's vector of Omega Network Hash Construction is taken from the input and the remaining column's vector is taken from XORed the previous column's vector, the parallel method is used to run parallel the proposed Omega Network Hash Construction while maintaining the security level as good as or better than SHA-512. Five designs of Omega Network Hash Construction (8, 16, 32, 64, 128 - for details, refer to Chapter 3) are simulated in this research to determine which size is more suitable for hash construction. Finally, the result of Omega Network Hash Construction SHA-512 is compared with the existing Merkle-Damgard construction SHA-512 to see whether there are any improvements in speed up, efficiency and overhead reduction.

1.7 Research Contributions

This research has produced an alternative architecture for hashing based on Omega Network Hash Construction. Chapter 3 explains in detail the architecture of Omega Network Hash Construction. Chapter 4 shows the performance comparison between three sizes of Omega Network Hash Construction with Merkle-Damgard construction. The proposed Omega Network Hash Constructions perform better than Merkle-Damgard construction.

1.8 Research Justification

Among the existing hash functions, SHA is chosen because of its popularity. Increasing the performance of SHA by changing the SHA algorithm requires a long and expensive development period. Thus, the more effective way to get better performance is by having an alternative architecture to the Merkle-Damgard construction. Omega Network is chosen due to its design can be executed parallel, the performance and speed up can be improved and the mix up output can provide better security in term of randomness.

1.9 Outline of the Thesis

This chapter shows an overview work for the research. The remaining Chapters are presented as below:

Chapter 2 discusses the related works and concepts which are useful for this research. An unkeyed primitive is one of the categories of cryptography primitives. The root architecture is Merkle-Damgard construction. The discussion in this chapter is based on Merkle-Damgard construction.

Chapter 3 shows the research methodology. Five phases of activities need to be followed to complete the research. The detail research design is also presented in this chapter.

Chapter 4 shows the performance and security results of Omega Network Hash Construction simulation. The performance is presented as graphs, and the security is shown as a table. The comparison of the performance and security between Omega Network Hash Construction and Merkle-Damgard construction is also discussed.

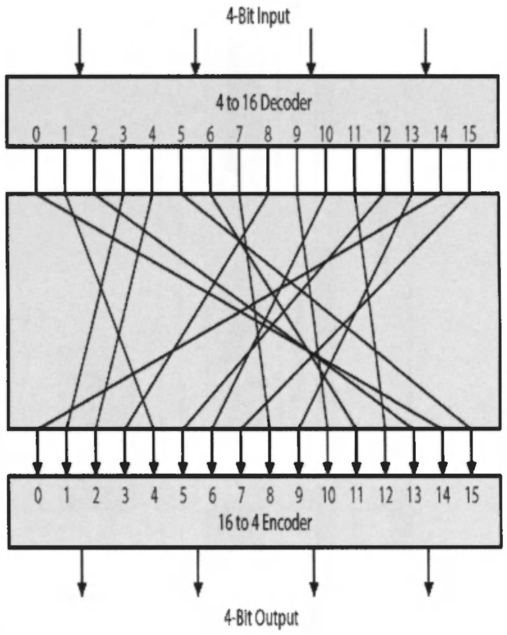
The conclusion and future work are presented in Chapter 5. The conclusion is reached through the result discussed in Chapter 4.

CHAPTER 2

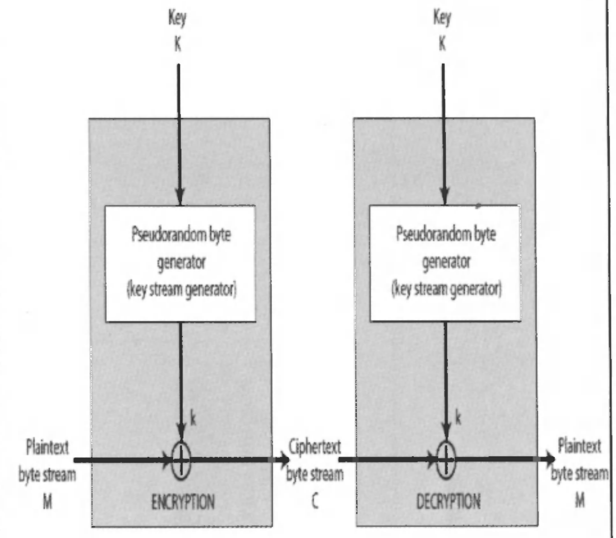
LITERATURE REVIEW

2.1 Cryptography

Cryptography can be classified into two major classes: public key and non-public key. For public key cryptography, the structure for encryption, key sharing and digital signature relies on mathematical properties such as finite fields (abstract algebra) and number theory (prime number, prime factorization, Fermat theorem, Euler theorem, etc). Non-public key cryptography includes secret key encryption and hash functions. Block cipher and stream cipher (see Figure 2.1) are two important architectures for secret key encryption. Merkle-Damgard construction (see Figure 1.1) is the root construction for all hashing functions. See Figure 2.2 for the classification of modern cryptography.

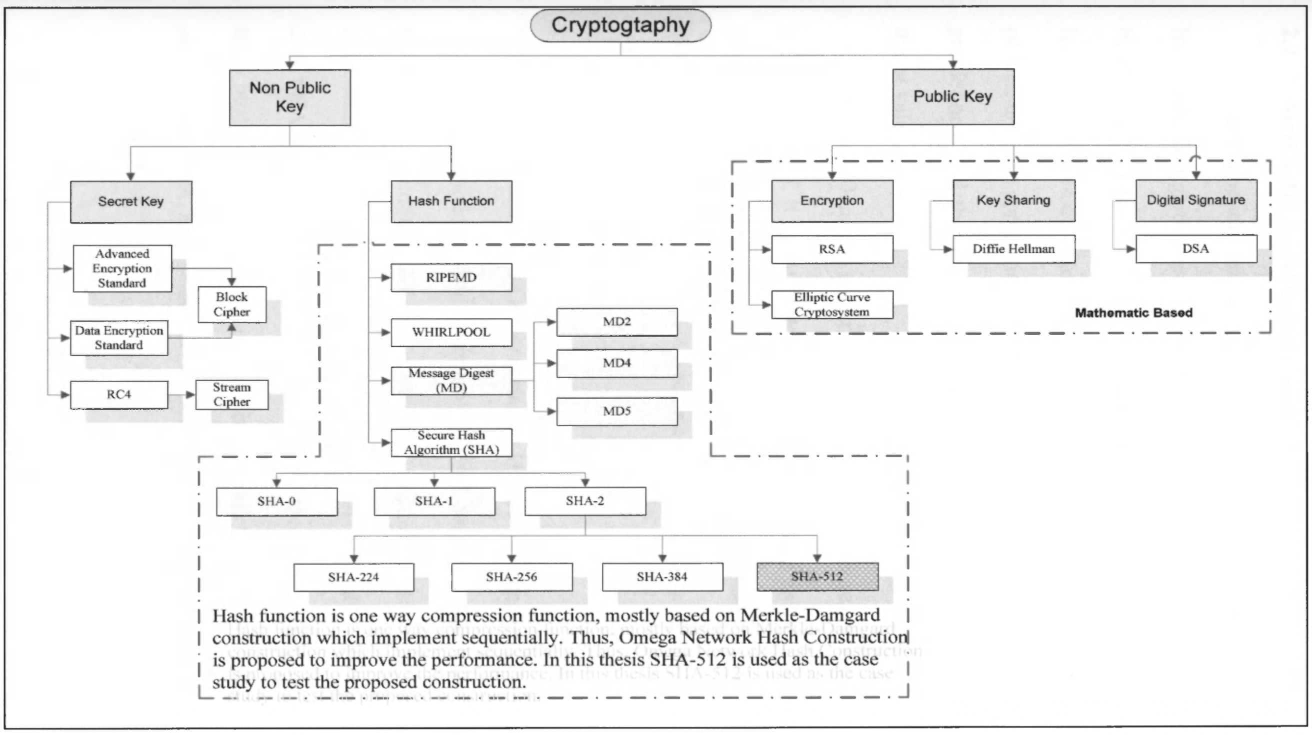


Block Cipher


Key
K
Key
K
Pseudorandom byte generator (key stream generator)
Pseudorandom byte generator (key stream generator)
k
k
Plaintext byte stream M
Ciphertext byte stream C
Plaintext byte stream M
ENCRYPTION
DECRYPTION

Stream Cipher

Figure 2.1: Two basic architectures for secret key encryption [30]



Hash function is one way compression function, mostly based on Merkle-Damgard construction which implement sequentially. Thus, Omega Network Hash Construction is proposed to improve the performance. In this thesis SHA-512 is used as the case study to test the proposed construction.

Figure 2.2: Cryptography classes

2.2 Merkle-Damgard Construction

Hash functions are based on the Merkle-Damgard construction (see Figure 1.1). In Merkle-Damgard construction, the input is broken up into a series of equal-sized blocks and processed in sequence with a one-way compression function. Normally the last block processed is an unambiguous size. To solve this problem, the last block is padded until the appropriate length is achieved. The vector for the first block function is pre-defined, and the remaining block function's vector is dependent on the previous digest value.

Merkle-Damgard construction runs over the input sequentially, and when the size of input is small, it performs well. But when the size of the input is large, the execution time will increase linearly. In terms of the security, it is based on the hash algorithm, for example the SHA family. SHA-2 is secure while SHA-0 and SHA-1 are not secure.

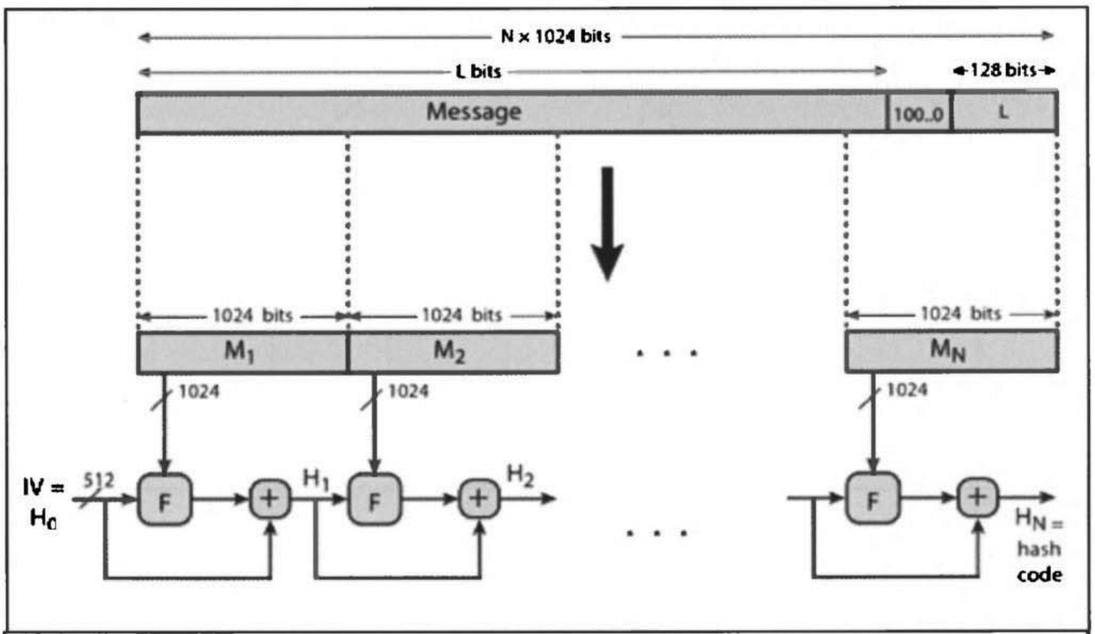
2.3 Merkle-Damgard Construction in Hash Functions

Hash functions ensure data integrity and data authentication. SHA family is one of the hash function based on the Merkle-Damgard construction. The members of the SHA family are SHA-0, SHA-1 and SHA-2. SHA-2 members include SHA-224, SHA-256, SHA-384 and SHA-512. The members of the SHA family are different based on their block size, their digest length, the number of constants that they used in the algorithm of SHA, and the number of iterations performed, (see Table 2.1).

Table 2.1: Functional characteristics of six hash functions [1, 21, 22, 30]

SHA Standard	SHA-0 (bits)	SHA-1 (bits)	SHA-224 (bits)	SHA-256 (bits)	SHA-384 (bits)	SHA- (bits)
Message digest size/ Size of hash value	160	160	224	256	384	512
Message size	$< 2^{64}$	$< 2^{64}$	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Message block size	512	512	512	512	1024	1024
Word size	32	32	32	32	64	64
No iteration in SHA algorithm	80	80	64	64	80	80
Number of constants	4	4	64	64	80	80

For easy reference, Figure 2.3 shows the Merkle-Damgård construction being used in SHA-512. The input or message is broken equally into 1024-bit blocks. If the last block is less than 1024 bits, the last block will be padded equal to 1024 bits. Each message block goes through the SHA-512 algorithm, which consists of 80 iterations and produces a 512-bit hash value.



References:
L bits – Total size of message in bits, IV – Initial vector

Figure 2.3: Message digest generation using SHA-512 [adopted from 11, 30]

2.4 Tree Based Merkle-Damgård Construction

National Institute of Standards and Technology (NIST) organized a competition for selecting SHA-3 currently (2009). Jason, W. M is the candidate for this competition; he submitted a paper called ESSENCE cryptographic hashing algorithm from which the construction for hashing algorithm is based on Tree Based Merkle-Damgård construction (see Figure 2.4). The ESSENCE design has been optimized using the parallel implementation and obtained better performance than the sequential Merkle-Damgård construction, but it had the poor performance on short messages. He claimed that his design is a modified version of the sequential Merkle-Damgård construction, thus his design share the same weaknesses with the existing construction. From the paper, there is no security test to show whether ESSENCE is secured.

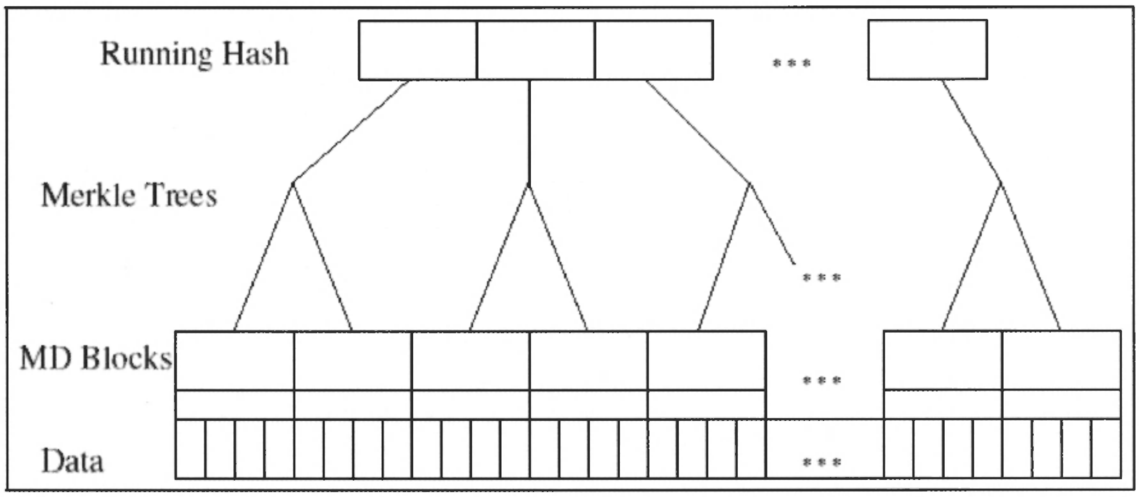
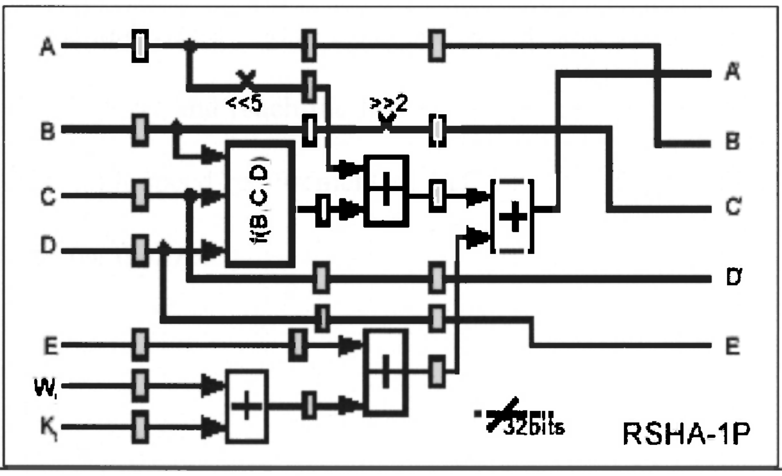


Figure 2.4: Overview of Tree Based Merkle-Damgård Construction [20]

2.5 Parallel Secure Hash Algorithm

Pipeline is one of the methods for function decomposition in the field of parallel. Pongyupinpanich and Choomchuay utilized the pipeline method to runs SHA-1, to enhance the performance of Digital Signature Algorithm (DSA) by overcoming unreasonable overhead in small applications [27]. There is small modification done on SHA-1, from which SHA-1 is coded to run in a pipeline mode (see Figure 2.5). $E + W_t + K_t$ is computed parallel with $A + B + C + D$. The authors claimed that the major drawback from this pipeline design is when there exist a higher number of pipeline states in which the design is cumbersome and the gate count increases dramatically. Thus, this pipeline SHA-1 with DSA is not scalable for all size application because it can only perform well in small size applications.



A, B, C, D, E – Block initialization. W_t – Word constants. K_t – Round constants.

Figure 2.5: An internal pipeline of a single round computational module [27]

2.6 Related Architecture for Hash Function

The Secure Hash Dynamic Structure Algorithm (SHDSA) [9] is used in many applications such as public key cryptosystems, digital signature, digital encryption, message authentication code and random number generators. All of these application's requirements are different from each other. As a result, Elkamchouchi's group proposed SHDSA which comes in a variety of configurations.

This dynamic scheme is based on SHA but with one major difference - the hash value is variable length with possible sizes of 128, 192, and 256 bits (see Figure 2.6). Besides that, the iterations in each function can be changeable based on the requirement of the applications. Thus, this dynamic scheme provides different levels of security for satisfying the choices for those practical applications.

Although SHDSA is designed to be changeable based on the requirement of the application, the architecture is formulated sequentially and the functions for SHDSA also executed sequentially (see Figure 2.7). The performance is affected; the execution time will increase linearly and reach the highest degree of throughput when the size of input is high. The high speed requirement of SHDSA is highly needed, which is why SHDSA should be parallelized.

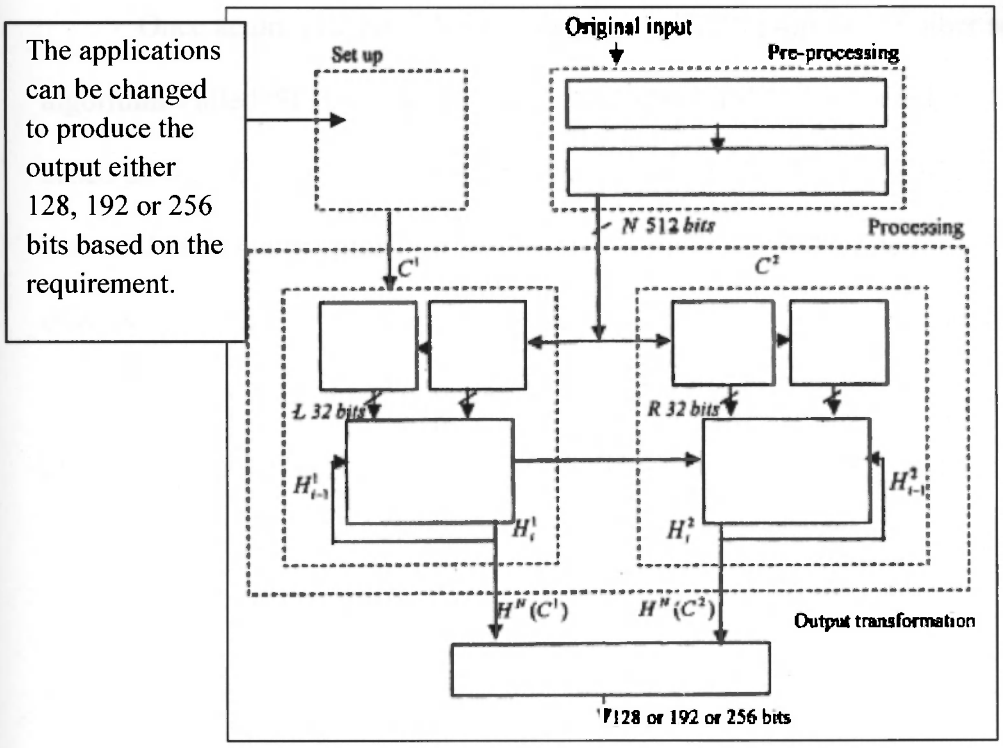

The applications can be changed to produce the output either 128, 192 or 256 bits based on the requirement.

Figure 2.6: The architecture of SHDSA [9]

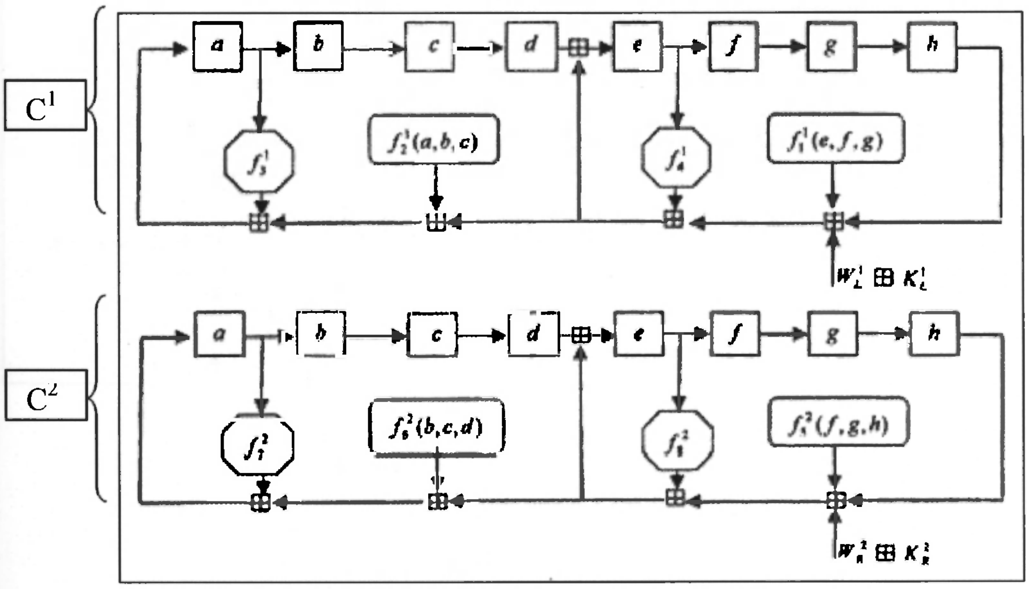


Figure 2.7: The sequential function in SHDSA, C^1 and C^2 [9]

Once again, in 2008, Elkamchouchi, *et. al.* [10] proposed another secure and fast algorithm called SFHA-256. This one was specifically designed for SHA-256. It is based on the 3C construction, which is based on the Merkle-Damgård construction. The author claims that the proposed architecture is more secure and performs better than the existing SHA-256. He claims that performance is better because the number of operations performed in a step function is reduced and because the architecture consists of two branches running in parallel (see Figure 2.8). SFHA-256 has fewer processing steps, but it is still secure because each step function contains operations that make it difficult for attackers to analyze SFHA-256. The added operations are simple XOR, addition and shift rotation operations. However, performance still suffers due to the waiting time that occurs during the processing of hash values.

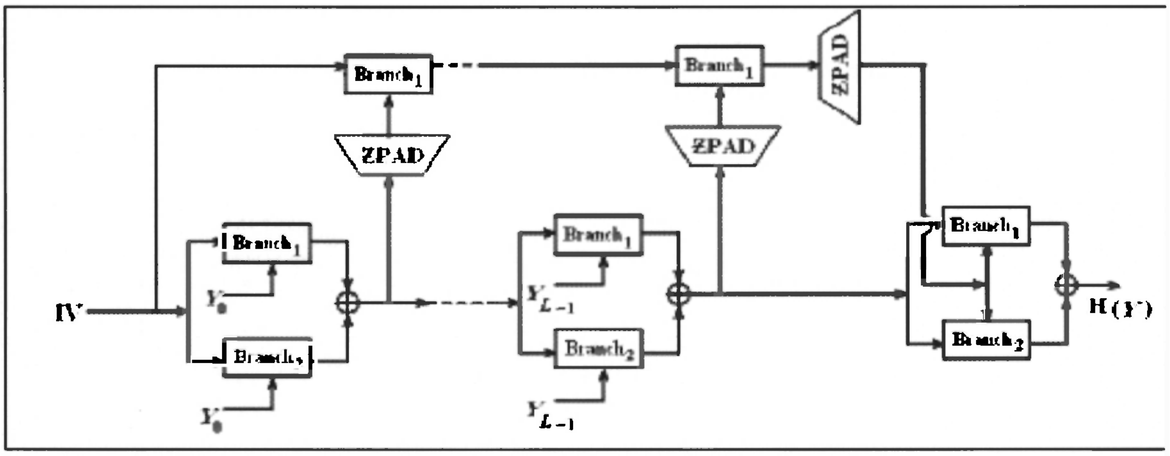


Figure 2.8: The architecture of SFHA-256 [10]

Gauravaram, *et. al.* [12] proposed the 3C+ hash construction which is based on the Merkle-Damgård construction. This 3C+ construction is an enhancement of 3C construction where a third internal chain has been added on top of the cascade and accumulation chains of 3C (see Figure 2.9). With this enhancement, the security level of 3C+ is better than both 3C construction and the Merkle-Damgård construction. 3C+ contains more XOR operations which also improves its security. However, in this new algorithm, there exist conditions where the hashing functions are required to wait for the input from the previous hash function. Moreover, the whole construction is sequential. Thus, waiting times can be extremely high in the 3C+ construction.

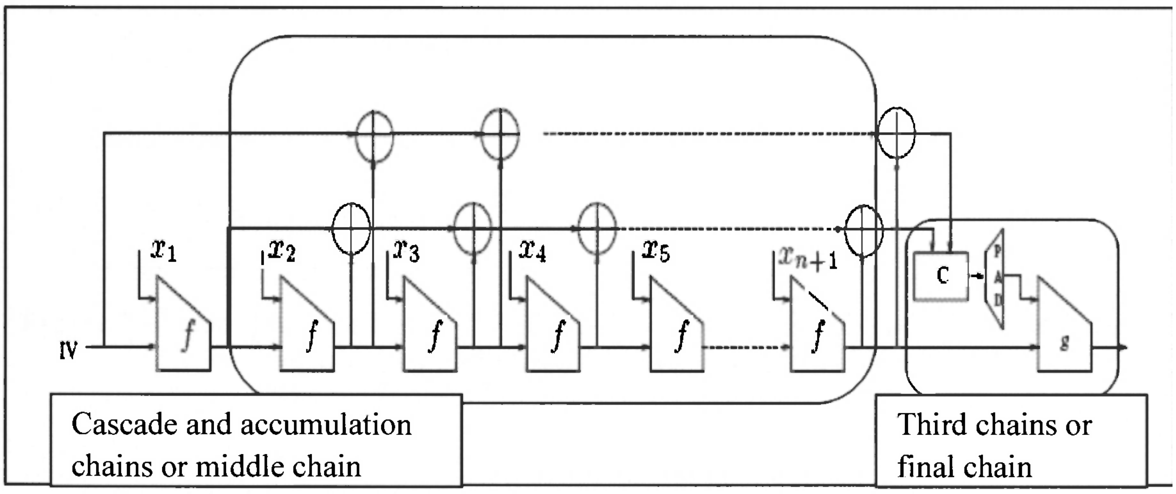
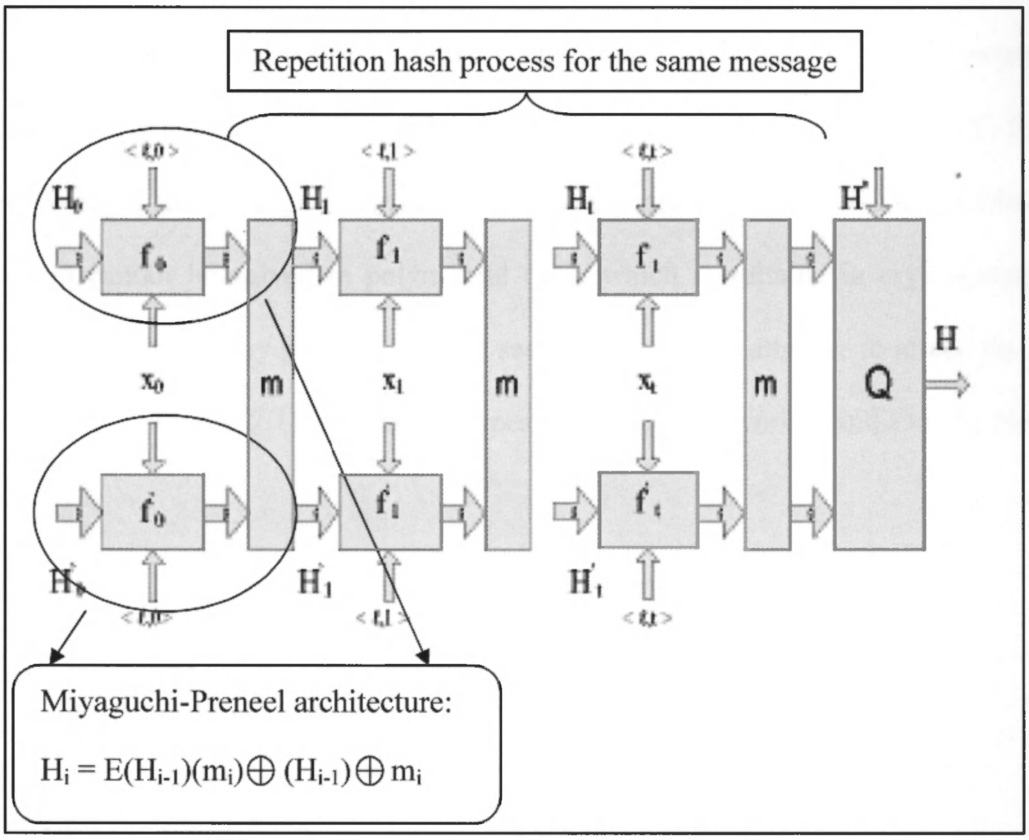

x_1
x_2
x_3
x_4
x_5
x_{n+1}
IV
f
C
PAD
g
Cascade and accumulation chains or middle chain
Third chains or final chain

Figure 2.9: The architecture of 3C+ Hash Construction [12]

Mirvaziri et. al. [21] came up with an enhancement to the Merkle-Damgård construction by developing a single-length compression function implemented on the Miyaguchi-Preneel block cipher. The architecture has intelligent repetition optimize hash process, which leads to better security (see Figure 2.10). Though the architecture is designed in double levels, it runs sequentially across the message, which means its computation time increases linearly when the input size increases.



Repetition hash process for the same message
<t,0>
H_0
f_0
x_0
f'_0
H'_0
m
H_1
<t,1>
f_1
x_1
f'_1
H'_1
m
H_t
<t,t>
f_t
x_t
f'_t
H'_t
m
H'
Q
H
Miyaguchi-Preneel architecture:
$H_i = E(H_{i-1})(m_i) \oplus (H_{i-1}) \oplus m_i$

Figure 2.10: Double-pipe Prefix-free Hash function with Miyaguchi-Preneel design resulted by Enhanced Merkle- Damgård [21]

In conclusion, most of the proposed architectures run sequentially, which means the computation time increases linearly when the input size increases. Given that multi-core technology and multithreaded programming are common in today’s world, these architectures are unacceptably slow.

2.7 Omega Network Construction

Omega Network is a construction widely used in telecommunication, due to its design that supports a switching function for the optical networks [4, 6, 28]. Beside that, the Omega Network also uses a network algorithm that is used in parallel computing architecture, which can run in multi-core machines to enhance performance. Yi Pan, *et. al* [26] urged that Omega Network is an NP-hard problem. An NP-hard problem is a problem cannot be solved in polynomial time, which is suitable in cryptography, for example, in hashing by providing better security to prevent attacker to attack the digest value easily. Figure 2.11 shows two types of Omega Network 8 and Omega Network 16.

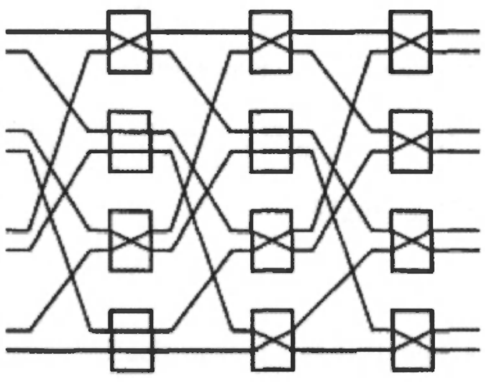
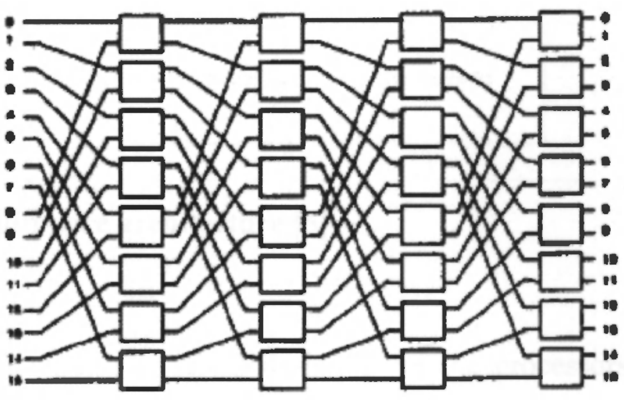
	
Omega Network 8 [28]	**Omega Network 16 [6]**

Figure 2.11: Omega Network's design

2.8 Multi-core Architecture

The multi-core architecture supports many simultaneous instruction executions via support of multithreaded programming. Intel and AMD processors designs implement coherent shared memory. The major difference is the design of level two caches. For the Intel processor, it is shared. For the AMD Dual Core Opteron, the level two caches are private for each processor (see Figure 2.12).

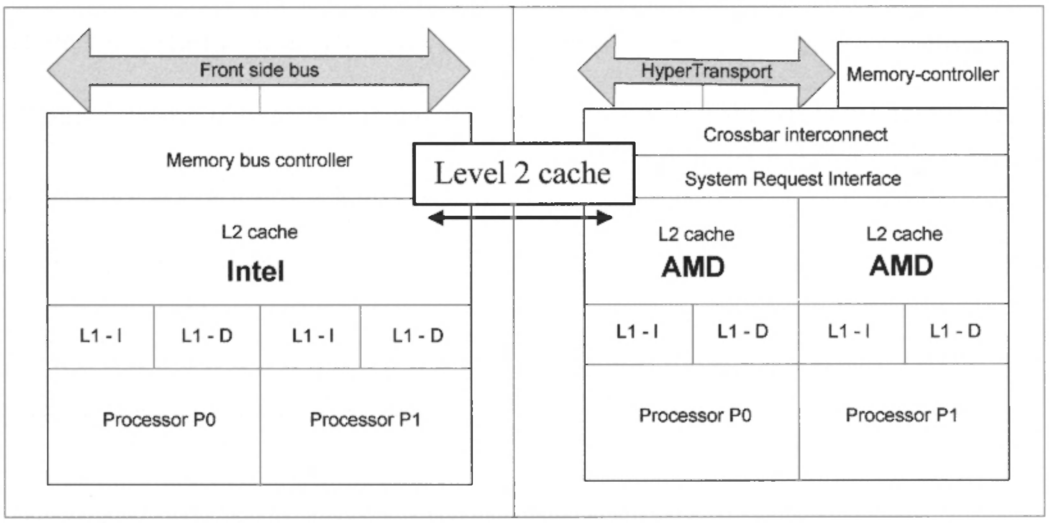


Figure 2.12: Multi-core architecture [5]

Each design has different advantages. The AMD design gives the processors more private memory and is therefore suitable for combining several dual processor chips. The Intel design allows the processors to use more than its share and support lower latency on-chip communication [5].

2.9 Parallel Model

To effectively achieve the better performance through the parallel computing, the parallel model plays an important role. There are two major parallel models: data parallelism and task parallelism.

2.9.1 Data Parallelism

Data parallelism is a parallel model where same functions or instructions are applied repeatedly on large data sets. There are two techniques that being used for distribute the data to the tasks static mapping and dynamic mapping (see Figure 2.13). The static mapping is a technique partition the data into number of blocks which equal to the number of processes. Dynamic mapping is a technique allocates subunits of the data to processes as and when they become free. These two techniques are varying in term of the computational progresses. Typically, OpenMP is used as the multithreading programming in expressing data parallelism [3, 5, 13, 14].

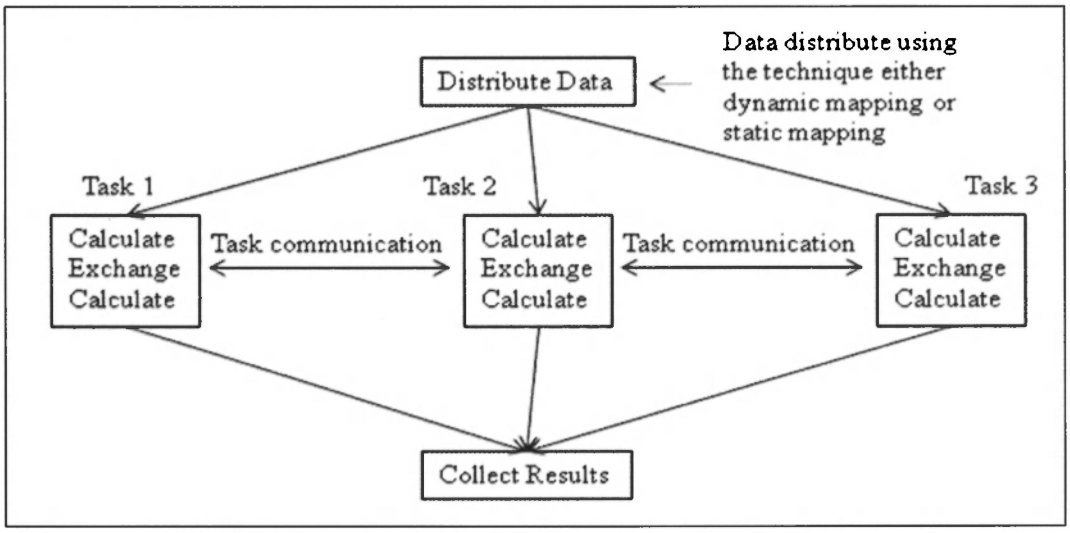

Data distribute using the technique either dynamic mapping or static mapping
Distribute Data
Task 1
Task 2
Task 3
Calculate Exchange Calculate
Task communication
Calculate Exchange Calculate
Task communication
Calculate Exchange Calculate
Collect Results

Figure 2.13: General overview for data parallelism [5, 13]

2.9.2 Task Parallelism

Task parallelism is a model that concurrently executes unique or independent tasks (see Figure 2.14). POSIX thread is a multithreading programming language to express task parallelism. The popular technique is pipelining and master/slave. Pipelining is a process that divides the task to be executed in the assembly line. Master/slave technique is the 'master' will distribute the task to be executed by 'slave' and then collect the result from 'slave', and then finally produce the final result [3, 5, 13, 14].

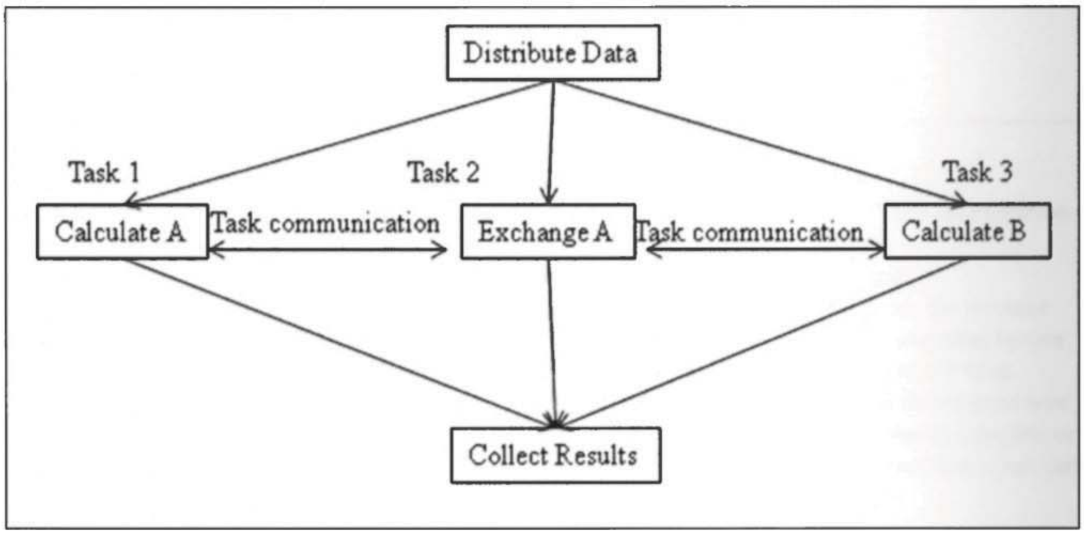


Figure 2.14: General overview for task parallelism [5, 13]

2.10 Multithreading Programming

The primary goal of multithreading programming is to enable concurrency and improve performance in applications. Two common multithreading programming languages are POSIX thread and OpenMP.

2.10.1 POSIX Thread

POSIX thread, commonly referred to as PThreads, is suitable for task parallelism. There are various types of POSIX thread methods, such as thread creation, termination, synchronization, and variable conditions (see Figure 2.15). These methods are used in ensuring the concurrency task in multi-core machine [5].

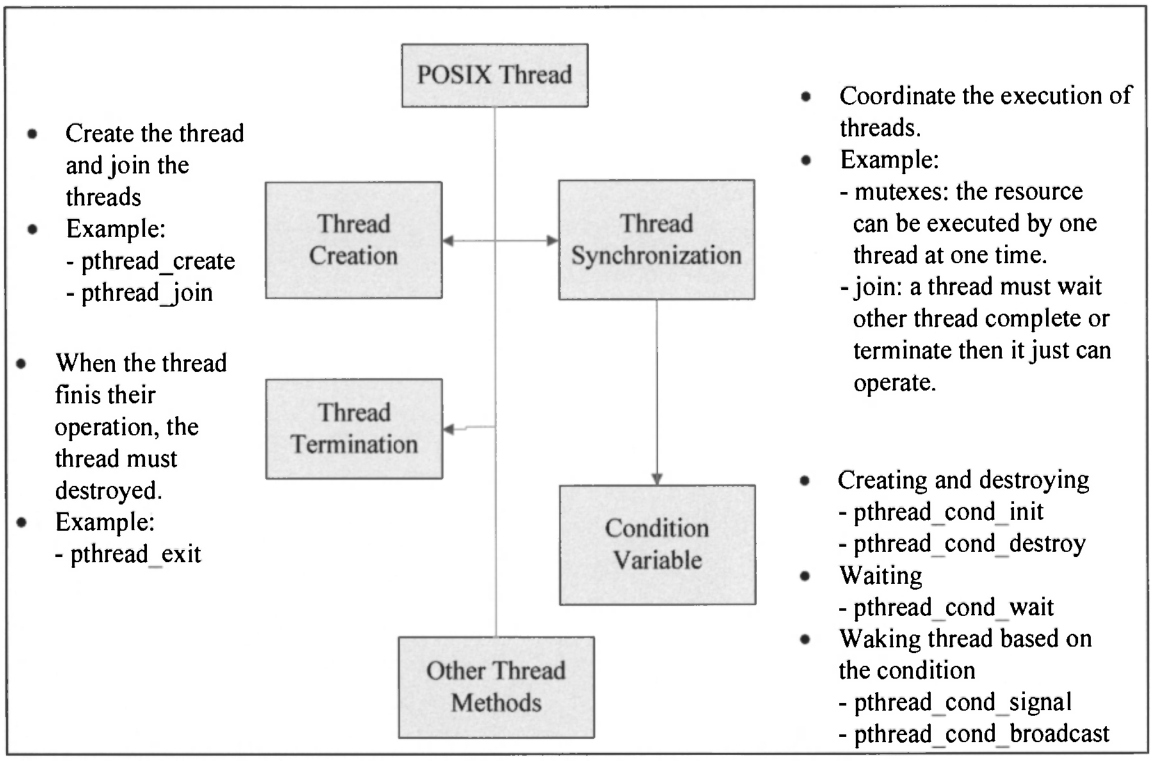

POSIX Thread
Thread Creation
Thread Synchronization
Thread Termination
Condition Variable
Other Thread Methods
• Create the thread and join the threads
• Example:
- pthread_create
- pthread_join
• When the thread finis their operation, the thread must destroyed.
• Example:
- pthread_exit
• Coordinate the execution of threads.
• Example:
- mutexes: the resource can be executed by one thread at one time.
- join: a thread must wait other thread complete or terminate then it just can operate.
• Creating and destroying
- pthread_cond_init
- pthread_cond_destroy
• Waiting
- pthread_cond_wait
• Waking thread based on the condition
- pthread_cond_signal
- pthread_cond_broadcast

Figure 2.15: POSIX thread parallel methods [5]

REFERENCES

[1] Ahmad, I., and Das, A. S. (2005). Hardware Implementation Analysis of SHA-256 and SHA-512 Algoritms on FPGAs. University Trier. Pp. 345 - 360.

[2] Ayguade, E., Copty, N., Duran, A., Hoeflinger, J., Lin, Y., Massaioli, F., Teruel, X., Unnikrishnan, P., and Zhang, G. S. (2008). The Design of OpenMP Tasks. In: IEEE Transactions on Parallel and Distributed Systems, Vol. 20, No. 3, March 2009. Pp. 404-418.

[3] Bal, H. E., and Haines, M. (1998). Approaches for Integrating Task and Data Parallelism. In: IEEE Concurrency, Vol. 6, No. 3. Pp. 74 - 84.

[4] Bernhard, P. **J.,** and Rosenkrantz, D. **J.** (1994). Partitioning Message Patterns for Bundled Omega Networks. In IEEE Transactions On Parallel and Distributed System, Vol. 5, No. 4. April 1994. Pp. 353 - 363.

[5] Calvin, L., and Snyder, L. (2009). Principles of Parallel Programming. Pearson International Edition.

[6] Das, S. and Chaudhuri, A. (1990). Analysis of the Effect of Size of Omega Network on its Fault Tolerance Behaviour in Presence of Multiple Faults. In: IEEE. Pp. 628-631.

[7] FDK Corporation. (2003). The Evaluation of Randomness of RPG100 by Using NIST and DIEHARD Tests. Pp 1 - 6.

[8] Kang, B. H., Lee, D. H., and Hong, C. P. (2008). Pseudorandom Number Generation Using Cellular Automata. © Springer Science+Business Media B.V. Pp.401-404.

[9] Elkamchouchi, H. M., Einarah, A. M., and Hagras, E. A. A. (2006). A New Secure Hash Dynamic Structure Algorithm (SHDSA) for Public Key Digital Signature Schemes. In: The 23rd National Radio Science Conference (NRSC 2006). Pp. 1 - 9.

[10] Elkamchouchi, H. M., Nasr, M. E., and Abdelfatah, R. I. (2008). A New Secure and Fast Hashing Algorithm (SFHA-256). In: 25th National Radio Science Conference (NRSC 2008). Pp. 1 - 8.

[11] Emam, S. A., and Emami, S. S. (2007). Design and Implementation of a Fast, Combined SHA-512 on FPGA. In: IJCSNS International Journal of Computer Science and Network Security, VOL. 7 No. 5, May 2007. Pp. 165 - 168. (IJCSNS).

[12] Gauravaran, P., Millan, W., Dawson, E., and Viswanathan, K. (2006). Constructing Secure Hash Functions by Enhancing Merkle- Damgard Construction. In: Springer - Verlag Berlin Heidelberg. Pp. 407 - 420.

[13] Grama. A., Gupta, A., Karypis, G., and Kumar, V. (2003). Introduction to Parallel Computing, Second Edition. Addison Wesley.

[14] Intel Corporation. (2003). Threading Methodology: Principles and Practices Version 2.0. Copyright © 2002, 2003 Intel Corporation. Pp. 15-42.

[15] Jackiewicz, M., and Kuriata, E. (2004). Analysis of Non-linear Pseudo-noise Sequences. In Springer - Verlag London, United Kigdom. Pp. 93 - 102.

[16] Jang, I. J and Yoo, H. S. (2006). Pseudorandom Number Generator Using Optimal Normal Basis. ICCSA 2006, LNCS 3982 ©Springer-Verlag Berlin Heidelberg. Pp. 206-212.

[17] Kim, S. J., Umeno, K., and Hasegawa, A. (2003). Corrections of the NIST Statistical Test Suite for Randomness. In: Chaos-based Cipher Chip Project, Presidential Research Fund, Communications Research Laboratory, Incorporated Administrative Agency. Pp. 1-14.

[18] Marsaglia, G. (1996). DIEHARD: A Battery of Test of Randomness. [Accessed 1st January 2009], Available from World Wide Web: http://i.cs.hku.hk/~diehard/cdrom.

[19] Marsaglia, G., and Tsang, W. W. (2000). Some difficult-to-pass tests of randomness. In: Innovation and Technology Support Programme, Government of Hong Kong, Grant ITS/277/0. Pp. 1 - 9.

[20] Martin, J. W. (2009). ESSENCE: A Candidate Hashing Algorithm for the NIST Competition. Pp. 1-43.

[21] Mirvaziri, H., Jumari, K., Ismail, M., and Hanapi, M. (2007). Collision Free Hash Function Based on Miyaguchi- Preneel and Enhanced Merkel- Damgard Scheme. In: The 5th Student Conference on Research and Development - SCOReD 2007 11 - 12 December 2007, Malaysia.

[22] NIST. (1993). Announcing the Standard for Secure Hash Standard, 180 - 1. In: National Institute of Standard. [Accessed 1st January 2009]. Available from World Wide Web: http://www.itl.nist.gov/fipspubs/fipl80-l.htm.

[23] NIST. (2002). Announcing the Standard for Secure Hash Standard, 180 - 2. In: National Institute of Standard. Pp. 1-75.

[24] NIST Special Publication 800-22. (2008). A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. In: National Institute of Standard. [Accessed 1st January 2009]. Available from World Wide Web: http://csrc.nist.gov/mg/.

[25] Olivar, G. (2007). FIPS 180-2 SHA-224/256/384/512 Implementation. [Accessed 1st January 2009], Available from World Wide Web: http: //www, ouah. or g/o gay/sha2 /

[26] Pan, Y., Ji, C. Y., Lin, X. L., and Jia, X. H. (2002). Evolutionary Approach for Message Scheduling in Optical Omega Networks. In: Proceedings of the Fifth International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP.02). Pp. 9-17.

[27] Pongyupinpanich, S., and Choomchuay, S. (2001). An Architecture for a SHA-1 Applied for DSA. In: Proceedings of the First Electrical Engineering/ Electronics, Computer, Telecommunications, and Information Technology (ECTI) Annual Conference. Pp. 133 - 136.

[28] Shen, X. J., Yang, F., Pan, Y. (1999). Equivalent Permutation Capabilities between Time Division Optical Omega Network and Non-optical Extra Stage Omega Network. Pp. 356-362.

[29] Shin, S. H., Park, G. D., and Yoo, K. Y. (2008). A Virtual Three-Dimension Cellular Automata Pseudorandom Number Generator Based on the Moore Neighborhood Method. In: Springer-Verlag Berlin Heidelberg. Pp. 174-181.

[30] Stallings, W. 2006. Cryptography and Network Security Principles and Practices. Prentice Hall.

[31] Sunar, B., and Koc, C. K. (2009). True Random Number Generators for Cryptography. Springer Science+Business Media, LLC. Pp. 55 - 72.