CHAPTER 2

SURVEY OF LITERATURE

This chapter explains the different methodologies of the point multiplication available today and also specifies the merits and demerits of each methodology in detail.

2.1 OVERVIEW OF POINT COMPUTATIONS

The addition and multiplication over modulo operation are two important operations in a finite field. The subtraction and division are calculated based on the inverse of addition and inverse of multiplication respectively (William Stallings 2006). These operations are mentioned as follows:

1. \[ c = (a+b) \mod p \in GF(p) \]

2. \[ c = (a \times b) \mod p \in GF(p) \]

3. \[ c = (a-b) \mod p \in GF(p) = (a+(-b)) \mod p \in GF(p) = (a+(\text{addition inverse element of } b)) \mod p \in GF(p) \]

4. \[ c = a/b \mod p \in GF(p) = a \times b^{-1} \mod p \in GF(p) \]
\[(a \times (\text{multiplication inverse element of } b) \mod p) \in GF(p)\]

where \(a, b, c, -a, a^{-1} \in GF(p)\) and \(p\) is a prime number.

Based on these rules, an Elliptic Curve over Finite Field is defined with two operations known as the point addition and the point multiplication. The point subtraction, point division and point squaring are solved through these operations with its inverse operations (Erkay Savas et al 2005). The rules of point addition and the point subtraction are as follows:

1. \(R(x,y) = P(x,y) + Q(x,y) \mod p\)
   
   It is a point addition which is solved by using Equation (1.7).

2. \(R(x,y) = P(x,y) + P(x,y) \mod p\)
   
   It is a point doubling which is solved by using Equation (1.6).

3. \(R(x,y) = P(x,y) - Q(x,y) \mod p\)
   
   \(R(x,y) = P(x,y) + (-Q(x,y)) \mod p\)
   
   \(R(x,y) = P(x,y) + Q(x,-y) \mod p\)
   
   \(R(x,y) = P(x,y) + Q(x,p-y) \mod p\)
   
   \(R(x,y) = (P(x,y) + \text{additive inverse of } Q(x,y))\)
   
   which is solved by using the Equation (1.7).

The second operation called point multiplication is the number of repeated point additions (Forouzan BA 2011). It is calculated based on the point doubling and point addition.

\[R(x,y) = k \times P(x,y) \mod p\]

\[= P(x,y) + P(x,y) + P(x,y) + \ldots + P(x,y) \mod p\]  \hspace{1cm} (2.1)
(It means that the k times of point additions)

\[ k = \sum_{i=1}^{\Sigma} P(x,y) \]

The point division is the product of point and its inverse values which is shown as follows:

\[ R(x,y) = \frac{P(x,y)}{Q(x,y)} \mod p \]

\[ = P(x,y) \times Q^{-1}(x,y) \mod p \]

\[ = P(x,y) \times \text{the multiplication inverse of } Q \mod p \]

The relationship between the finite field operations and the Elliptic Curve equation is shown in Figure 2.1.

\[ \text{Figure 2.1 Relationship among the different operations on EC over GF (p)} \]
The point multiplication over EC is implemented in several ways. Some of the methodologies are explained in the following sections to understand the memory requirements and the speed. For this purpose, the different sections are organized as follows: The section-2.2 gives the overall view of different point multiplication methodologies, then, it describes linear point multiplication, double-add point multiplication, Montgomery point multiplication and Jacobian point multiplication in the different respective sections of 2.3, 2.4, 2.5 and 2.6.

2.2 POINT MULTIPLICATION METHODOLOGIES

Basically, a point addition is an atomic operation because the remaining operations are computed based on this operation. When it is implemented with a software routine, the execution time of this operation is a basic execution time of EC over finite field operations. At the same time, the point multiplication is a complex operation which needs more execution time to find out its results. It is defined with modulo, finite field and polynomial concepts (Bartolini S et al 2007). So it is also called as modular multiplication over finite field for polynomial. The time complexity of EC over finite field is mainly derived based on this operation. The point multiplication is classified into three types: General Moduli, Special Moduli and Lookup Table methods (Jaewook Chung et al 2007). The examples for General Moduli are Classical, Barrett and Montgomery algorithms, Modular Reduction methods based on Pseudo-Mersenne, Generalized Mersenne numbers, which are examples for Special Moduli. Finally, the examples for Lookup Table methods are Kawamura et. al., Hong et. al., and Lim et. al. methods.

The look table methods need the extra space for storing the intermediate values of point multiplications. It is not necessary to compute repeated point operations once again. The speed of this point multiplication is increased and the time complexity will become better than other types of
point multiplications (Wide W Waf et al 2007). The space complexity of this type is not better than others (Schmidt-Samoa K et al 2006). It is only suitable for a special type of parameters which supports fixed values of elliptic curve (Ming Li et al 2007). Hence, it is also called as fixed point multiplication.

The special moduli multiplications are examples for random point multiplication. Most of the parameters in this operation are variables and it is suitable for any type of Elliptic Curve. One of the special moduli is a Generalized Mersenne based on the mersenne number and finite field. This number is defined in the form of $2^n-1$ and it is used for modular reduction. Some of the mersenne numbers are not suitable for cryptographic applications, because they are composite numbers. If the value of ‘n’ is four, it will be 15 (composite). So the form of $2^n-1$ is changed into $2^n-c$, where c is a small integer. This type of mersenne number for modular reduction is known as pseudo mersenne. But it is also not suitable for cryptosystems because of integer factorization (Sudha KR et al 2007).

The classical algorithm of general moduli is based on the binary sequence of value and its weight. It is generally called doubling and adding methodology. It is easily attackable based on the side channel attack (Xiaoyu Ruan2005). The Montgomery of classical algorithm is better than this methodology, because the side channel attack is avoided. The time complexity of this is not better than the double and add methodology (Koç CK 1996). Another example for classical algorithm is a Barrett algorithm which is used to reduce the number of inversion computations through pre-computations. It needs a small amount of memory for storing these results. But the time complexity of this method is better than montgomery multiplication.
These point multiplications are analyzed with the help of computational complexity for the following reasons (Dromey RG 2007):

- Predict the resources needed for the algorithm
- Computational speed
- Memory space
- Communication bandwidth consumption
- Algorithm complexity (the total number of primitive operations during the execution)
- Crypto security based on mathematics
- Available implementations for different applications

A computational complexity is a mathematical characterization of a problem. It describes the resources needed for computing machine to solve the problem. It is more important for many branches of theoretical computer science. Especially, it is in cryptography to analyze this complexity in the form of time complexity for processor speed and space complexity for memories (Sunar B 2005).

Based on this, the space complexity and time complexity are defined as follows:

“How much memory space are needed to solve the problem”

“How much time is required to respond the problem for solving”.

This computational complexity is normally classified into three types such as best, worst, and average cases.
Worst-case time complexity:

It is a function \( T(N) \) defined with the maximum number of steps taken on any function \( F(N) \) to execute problem with size \( n \).

Best-case time complexity:

It is a function \( T(N) \) defined with the minimum number of steps taken on any function \( F(N) \) to execute problem with size \( n \).

Average-case time complexity:

It is a function \( T(N) \) defined with some steps which is greater than minimum and lesser than maximum taken on any function \( F(N) \) to execute problem with size \( n \).

The corresponding ways of the notation, mathematical expression and relative growth rate of each time complexities are mentioned in the following Table 2.1 (Mark Allen Weiss 2007).

Table 2.1: Time complexity of worst, average and best cases notations, mathematical expressions and relative growth rates of the general problem

<table>
<thead>
<tr>
<th>Type</th>
<th>Notation</th>
<th>Mathematical Expression</th>
<th>Relative growth rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>worst case</td>
<td>Big-O</td>
<td>( T(N) = O(F(N)) )</td>
<td>( T(N) \leq F(N) )</td>
</tr>
<tr>
<td>best case</td>
<td>Big-( \Omega )</td>
<td>( T(N) = \Omega(F(N)) )</td>
<td>( T(N) \geq F(N) )</td>
</tr>
<tr>
<td>average case</td>
<td>Big-( \theta )</td>
<td>( T(N) = \theta F(N) )</td>
<td>( T(N) = F(N) )</td>
</tr>
</tbody>
</table>

For improving the speed of the algorithm, the parallel computation is implemented to execute the coding in concurrently. In this case, it is
necessary to identify different hazards during or before the executions based on the dependencies and it is tried to eliminate them. These hazards are created based on read or write operations. The hazards are classified into four types such as: Read After Read, Read After Write, Write After Read and Write After Write.

For example, Two events i and j are considered. Here i\textsuperscript{th} event is occurring before j\textsuperscript{th} event. The RAW (read after write) is defined as: the j\textsuperscript{th} event tries to read a source before the i\textsuperscript{th} event writes the result, so the j\textsuperscript{th} event incorrectly gets the result. The WAW (write after write) is defined by: Event j tries to write an operand before it is written by Event i. The writing is performed in the wrong order, then it leaves the value written by event i rather than the value written by event j in the destination. The third WAR (write after read) is an event j to try for writing on the destination before it is read by an event i. So the event i incorrectly gets the new value. Finally RAR (Read After Read) is not a hazard, because it is frequently reading the same value of corresponding event i or j (Hennessy JL 2007).

It is necessary to find out the suitable implementation for point multiplication which avoids these hazards to create a possibility for parallel processing (Dromey RG 2007). So, the following rules are applied to schedule coding for the point multiplication on ECC or ECDSA.

1. Avoid repeated computations as much as possible
2. Try to simplify array reference
3. Find out desired output earlier
4. Try to avoid late termination in the iteration
5. Maintain the minimum amount of memory space as much as possible with time
Mostly, the algorithm is designed with the top down or bottom up design. Here, the point multiplication problem is divided into a number of sub problems such as the point addition or the point doubling. Then it chooses a suitable data structure to solve the number of point additions and point doublings separately. Besides, the initial condition, terminating loop and implementing iterative construct of the point multiplication are properly mentioned to avoid early termination, late detection and redundancy repetitions. The survey of literature is also conducted to identify the different merits and demerits of the point multiplication for better enhancements.

2.3 SCALAR POINT MULTIPLICATION

The scalar point multiplication is an operation to calculate a multiple times of an element based on the finite field. Similarly the point multiplication is an operation to calculate a multiple times of a point addition over finite field as shown in Equation (2.1). It is known as linear scalar point multiplication. First, it is computed based on the point doubling one time and it is followed the ‘n-2’ times of point additions (Komathy K et al 2006).

2.3.1 Overview

The scalar point multiplication of \( kP \) is the result of adding \( k-1 \) times of the point value \( P \) to itself.

\[
k.P(x,y)=(P(x,y) + P(x,y) + P(x,y) + \ldots + P(x,y)) \mod p \in GF(p)
\]

point doubling

\[
= 2^1 P(x,y) + P(x,y) + \ldots + P(x,y) \mod p \in GF(p)
\]

point addition
\[(k-1)P(x,y) + P(x,y) \mod p \in \text{GF}(p) \]

point addition

\[= kP(x,y) \in \text{GF}(p)\]

So the \(kP(x,y)\) is computed with a point doubling and the number of point additions and is shown in Figure 2.2.

Figure 2.2 Scalar point multiplication of \(kP(x,y)\) where \(k\) is a scalar value and \(P(x,y)\) is a point value

The implementation of this scalar point multiplication is given in the form of procedure as follows:

\[\text{procedure linearmultiplication()}\]

Input: Point \(P(x,y)\), Integer \(k\).

Output: \(kP(x,y)\)

Integer \(I\);

Point \(Q(x,y)=(0,0)\);

1. \(I=1;\)

2. repeat

3. if \((P(x,y)== Q(0,0))\) then // based on Equation (1.4)

4. \(Q(x,y)=P(x,y);\)

5. else if\((P(x,y)== Q(x,-y))\) then // based on Equation (1.5)
6. \( Q(x,y) = P(x,0); \)
7. else if \( (P(x,y) == Q(x,y)) \) then // based on Equation (1.6)
8. \( Q(x,y) = P(x,y) + Q(x,y); \)
9. else // \( (P(x,y) != Q(x,y)) \) // based on Equation (1.7)
10. \( Q(x,y) = P(x,y) + Q(x,y); \)
11. end if
12. \( I = I + 1; \)
13. until \( (I < k); \)
14. \( kP(x,y) = Q(x,y) \)
15. return \( kP \)

2.3.2 Limitation

This multiplication has two important demerits. One is the time complexity and another is no possibility of parallelism. Normally, the time complexity of linear multiplication denoted by \( O(n) \) is worst case, because the process is repeated \( (n-1) \) times of point additions (Bertoni G et al 2006).

The second is that there is no possibility for parallel computation, because there are different types of dependencies such as: data, name, control and loop carried dependencies existing in the linear point multiplications (Jarvinen K et al 2008). Here, the name dependency means that two or more points refer to the same name. In this multiplication, a point addition and a point doubling share the common variable to update the point value in each time. Next, a point is dependent on another point known as data dependencies. For example, the point addition or point doubling is always dependent on earlier point addition or point doubling.
A control dependence determines the way of computing points based on four constraints mentioned in Equations (1.4), (1.5), (1.6) and (1.7). In this point multiplication, a point addition or point doubling is always dependent on the value of previous point addition or a point doubling. The way of determining execution path (point addition, point doubling) is an example for control dependencies. Finally, the loop-carried dependence is carried out from the current point computation to next point computation in each time in the loop. In linear point multiplication, a point addition or point doubling always depends on the previous processes. The merit of point multiplication is a simple and straightforward computation.

2.4 DOUBLE AND ADD POINT MULTIPLICATION

The Double and Add Point Multiplication is computed, based on the scalar value (k) through scanning of bits. Here, the point doubling is computed compulsorily and the point addition is only calculated depending upon the bit value (Kenny Fong et al 2004). If the value of scalar bit is ‘1’, the point addition will be performed.

2.4.1 Overview

The Double and Add Point Multiplication depends upon the binary length of scalar value to perform the number of point additions. The point addition may be only a point doubling or a point doubling with a point addition (Kumar S et al 2006).

\[ k = \text{the number of times} \]
\[ n = \text{the binary length of } k \]

So, \( k \cdot P(x,y) = (n-1) \text{ times of point doubling} + \text{the number of point additions based on the value of } n \) where \( n = 1 \)
\[ n-1 = \sum_{i=1}^{n-1} (P_i(x,y) + P_i(x,y)) + a_i (P_i(x,y)) \mod p \in GF(p) \]

where \( i \) is a position of binary digit in \( n \) varies from 1 to \( n-1 \) and \( a_i \) is a binary value of \( i^{th} \) digit which is equal to 1.

It is implemented in two different ways. One is the left to right binary methodology and another is right to left binary methodology (Xiaeya Ruan et al 2005). The way of point computation is diagrammatically shown in Figure 2.3.

![Diagram](image)

**PD → Point Doubling and PA → Point Addition**

**Figure 2.3** Scalar point multiplication of \( k.P(x,y) \) where \( n \) is a binary value of \( k \) assumed as 5 and \( P(x,y) \) is a point value

The corresponding coding is explained in the part of pseudo code 2.3 and pseudo code 2.4.

```plaintext
// pseudo code for RighttoLeft scalar point multiplication-1

Procedure RighttoLeft( )
```
Input: Point P(x,y),
Output: kP(x,y)
Integer i,n
Integer k(k_{n-1}, \ldots, k_0)_2
Point R_0(x,y), R_1(x,y)

1. R_0(x,y) = P(x,y)
2. R_1(x,y) = P(x,y)
3. i = n
   3.1. if (n!=0) then
       3.1.1 R_0(x,y) = R_0(x,y) + R_0(x,y)
       3.1.2 if k_i=1 then
           3.1.2.1 R_0(x,y) ← R_0(x,y) + R_1(x,y)
   3.2. k = k+1
   3.3. n = one time shift right of n and goto step 3.1

4. kP = R_0(x,y)
5. return kP

// pseudo code for LefttoRight scalar point multiplication-2

Procedure LefttoRight()

Input: Point P(x,y),
Output: kP(x,y)
Integer i,n
Integer k(k_{n-1}, \ldots, k_0)_2
Point R_0(x,y), R_1(x,y)
1. \( R_0(x,y) = 0 \)

2. \( R_1(x,y) \leftarrow P(x,y) \)

3. \( i = 1 \)

   3.1. \( \text{if( } n \leq k \text{) then} \)

      3.1.1. \( \text{if } k_i = 1 \text{ then} \)

      3.1.1.1 \( R_0(x,y) \leftarrow R_0(x,y) + R_1(x,y) \)

      3.1.2. \( R_1(x,y) \leftarrow R_1(x,y) + R_1(x,y) \)

3.2. \( i = i + 1 \)

3.3. \( n = \text{one time of shift left of } n \text{ goto step 3.1} \)

4. \( kP = R_0(x,y) \)

5. \( \text{return } kP \)

2.4.1 Limitation

This multiplication tremendously reduces the number of point additions. The number of point doublings and point additions are performed based on the value of binary digits \( (n_i = 1) \) in sequence. This sequence is scanned from least significant bit (LSB) to most significant bit (MSB) or MSB to LSB. The time complexity of this point multiplication is better than the previous point multiplication. It is denoted by \( O(\log_2 m) \) where \( m \) is the number of point additions (Chia-Long Wu et al 2005).

The second limitation is that it does not support for parallel computation, because it also has the same types of above mentioned dependencies. But the numbers of dependencies are lesser than the previous methods. The main weakness of this methodology is the side channel attack. Here, the scalar bit values are traceable based on the power analysis (Chevallier-Mames B et al 2004). Besides, these algorithms are only suitable
for special type of elliptic curve such as Montgomery Curve.

2.5 MONTGOMERY POINT MULTIPLICATION

The montgomery approach is a methodology to compute the point multiplication without side channel attack. This can be beneficial when the timing or power consumption are exposed to an attacker for performing a side channel attack (Page D et al 2004).

2.5.1 Overview

The point multiplication is similar to the same representation of double-and-add. But each time it performs the point addition followed by point doubling or vice versa. So it does not leak any information against the timing or power analysis to attackers (Ming-Der Shieh 2010). It is suitable for any type of EC coordinate system (O' Rourke C et al 2003). This point multiplication processes a loop scanning of bits ‘k’ and performs a point addition and point doubling. It depends upon the binary length of k value denoted by n and performs n-1 times of point additions as well as point doubling (Savas E et al 2000).

\[ n = \text{the binary length of } k \]

So, \( k \cdot P(x, y) = (n-1) \text{ times of point addition} + (n-1) \text{ times of point doubling} \)

\[ n-1 \]
\[ = \sum_{i=1}^{n-1} (P_i(x, y) + P_i(x, y)) + 2P_i(x, y)) \mod p \in GF(p) \]

where \( i \) is a position of binary digit in \( n \) varies from 1 to \( n-1 \) and \( a_i \) is a binary value of \( i^{th} \) digit which is equal to 1.
This type of point multiplication is diagrammatically shown in Figure 2.4 for better understanding.

![Diagram of Montgomery point multiplication]

**Figure 2.4  Montgomery point multiplication of kP where k is 5 and P is a point value**

**Procedure Montgomery_point_multiplication()**

Input: Integer k>0, P (x,y), i

Output: Integer Q=kP(x,y)

Integer k←(k_{n-1},...,k_1,k_0)_2

P_1(x,y)=P(x,y),
P_2(x,y)=2P(x,y)
i=n-2

repeat

if k_i=1 then

P_1(x,y)=P_1(x,y)+P_2(x,y)
P_2(x,y)=2P_2(x,y)

else
\[ P_2(x,y) = P_2(x,y) + P_1(x,y) \]
\[ P_1(x,y) = 2P_1(x,y) \]
until \( i \geq 0 \)
return \( Q(x,y) = P_1(x,y) \)

2.5.2 Limitation

This multiplication removes one of the demerits in the previous multiplication known as the side channel attack. It means that it computes a point addition and a point doubling in each time in the loop. So it is not possible to assume whether it is point addition or point doubling. But the time complexity of this point multiplication is not better than the double-add point multiplication (Majid Khabbazian et al 2007). It is computed based on the time complexity of point addition \( O(\log_2 m) \) as well as the time complexity of point doubling \( O(\log_2 m) \) (Huapeng Wu 2002).

Another limitation is the possibility of parallel computation. It also does not support parallel processing because of dependencies. In this point multiplication, both point doubling and point additions of current iteration are always dependent on the previous point doubling and point addition based on branch condition. There is a continuous data dependency between the point addition and the point doubling. It is the best example for chain data dependency which is available throughout the point multiplication (Tenca AF et al 2003).

2.6 JACOBIAN POINT MULTIPLICATION

Finally, the Jacobian point multiplication is a regular binary point multiplication which requires more field registers for storing the inverse value of point (Adnan Abdul-Aziz Gutub2010). The number of data dependencies,
register dependencies, and control dependencies are increased because of maintaining intermediate values. But there is no change in the loop-carried dependencies and branch dependencies.

2.6.1 Overview

This point multiplication depends upon the different point coordinate system to avoid inversions operations. A coordinate system means that it is an organization with one or more coordinates to distinctively determine the position of a point. The use of a coordinate system allows the problems in geometry to be translated into numbers, and numbers into geometry points. An elliptic curve can be represented with different coordinate systems. The point addition and point doubling can be implemented based on these coordinate systems. Some of the coordinates are as follows:

- Affine coordinate system
- Standard projective
- Standard projective and affine
- Jacobian projective
- Jacobian projective and affine
- Lopez-Dahab

The affine coordinates system is a normal form of elliptic curve without any projection to produce the value by using the line equation $y = mx + c$, where $m$ is the slope and $c$ is the $y$-intercept. It needs field inversion for both point addition and point multiplication. The point inversions are very exclusive that can be removed by changing the representation of the points (Jithra Adikari et al. 2008). For example, the affine coordinate $(x, y)$ is changed into the projective coordinate format $(X, Y, Z)$ where $x = (X/Z)c$ and $y =
(Y/Z)D. In this case C and D are multiplier parameters.

Some of the examples for projective coordinate systems are Projective, Jacobian, and Lopez-Dahab system which do not require an inversion operation. But it needs the additional information to make changes in the coordinate systems. For example, the projective coordinate system is represented by \((X,Y,Z)\) in three coordinates where \(x = X/Z\) and \(y = Y/Z\). The second example is Jacobian system which is represented with three coordinates \((X,Y,Z)\) where \(x = X/Z^2\) and \(y = Y/Z^3\). The third example is López–Dahab coordinate system which is defined by \(x = X/Z, \ y = Y/Z^2\). The fourth example is Chudnovsky Jacobian system. It consists of five coordinates in the form of \((X,Y,Z,Z^2,Z^3)\). Besides, the IEEE P1363-2000 standard is a projective coordinate with Jacobian coordinate model. In mathematics, the Jacobian coordinates system is used for to represent an Elliptic Curve which is different from Equation (1.1) and it is faster than Weierstrass curve (Darrel Hankerson et al 2000).

In homogeneous projective coordinates, each affine point \((x,y)\) is represented by three coordinates \((X,Y,Z)\) where \(x = X/Z\) and \(y = Y/Z\). Another coordinate system that is widely used in practice is the Jacobian projective coordinate system. Here, the relation \(x = X/Z^2\) and \(y = Y/Z^3\) is used to represent the points. The curve equation in Jacobian coordinates becomes \(E : Y^2 = X^3 + aXZ^4 + bZ^6\) (Lauter K 2004). For example, the \(P_1\) and \(P_2\) are assumed as \((X_1,Y_1,Z_1)\) and \((X_2,Y_2,1)\) respectively in the Jacobian projective coordinate. The sum of \(P_1 + P_2\) is computed and is denoted by \((X_3,Y_3,Z_3)\). The \(X_3, Y_3\) and \(Z_3\) are defined as follows:

\[
X_3 = (Y_2Z_1^3-Y_1)^2 - (X_2Z_1^2-X_1)^2(X_1 + X_2Z_1^2)
\]

\[
Y_3 = (Y_2Z_1^3-Y_1)(X_1(X_2Z_1^2-X_1)^2-X_3) - Y_1(X_2Z_1^2-X_1)^3
\]

\[
Z_3 = (X_2Z_1^2-X_1)Z_1.
\]
The different variables A, B, C, D, E, F, G, I, J, L, M, N, O, P, Q and R are used to maintain the intermediate values of this coordinate system for the point multiplication.

\[ A = Z_1^2 \]
\[ B = A.Z = Z_1^3 \]
\[ C = B.Y_2 = Y_2Z_1^3 \]
\[ D = C – Y_1 = Y_2Z_1^3 – Y_1 \]
\[ E = D^2 = (Y_2Z_1^3 – Y_1)^2 \]
\[ F = X_2A = X_2 Z_1^2 \]
\[ G = F – X_1 = X_2 Z_1^2 - X_1 \]
\[ H = G^2 = (X_2 Z_1^2 - X_1)^2 \]
\[ I = X_1 + F = X_1 + X_2 Z_1^2 \]
\[ J = HI = (X_2 Z_1^2 - X_1)^2(X_1 + X_2 Z_1^2) \]
\[ K = E – J = (Y_2Z_1^3 – Y_1)^2 – (X_2 Z_1^2 - X_1)^2(X_1 + X_2 Z_1^2) = X_3 \]
\[ L = HX_1 = X_1(X_2 Z_1^2 - X_1)^2 \]
\[ M = L - X_3 = X_1(X_2 Z_1^2 - X_1)^2 - X_3 \]
\[ N = D.M = (Y_2Z_1^3 – Y_1)(X_1(X_2 Z_1^2 - X_1)^2 - X_3) \]
\[ O = H.G = (X_2 Z_1^2 - X_1)^3 \]
\[ P = OY_1 = (X_2 Z_1^2 - X_1)^3 Y_1 \]
\[ Q = N – P = (Y_2Z_1^3 – Y_1)(X_1(X_2 Z_1^2 - X_1)^2 - X_3) – (X_2 Z_1^2 - X_1)^3 Y_1 = Y_3 \]
\[ R = GZ_1 = Z_1(Y_2Z_1^3 – Y_1)(X_1(X_2 Z_1^2 - X_1)^2 - X_3) – (X_2 Z_1^2 - X_1)^3 Y_1 = Z_3 \]

The point addition is evaluated based on these point computations. Here, this point computation cost is measured as the sum of 3 point squaring,
8 point multiplication, 5 point subtraction and 1 point addition without point inversion of $P_1+P_2$.

Similarly the point doubling is denoted by $2P_1 = (X_4, Y_4, Z_4)$ and it is calculated by:

\[
\begin{align*}
X_4 &= (3X_1^2 + aZ_1^4)^2 - 8X_1 Y_1^2 \\
Y_4 &= (3X_1^2 + aZ_1^4)(4X_1 Y_1^2 - X_3) - 8Y_1^4 \\
Z_4 &= 2Y_1 Z_1 \\
\end{align*}
\]

This case,
\[
\begin{align*}
A &= X_1^2 \\
B &= 3A = 3X_1^2 \\
C &= Z_1^2 \\
D &= C^2 = Z_1^4 \\
E &= D.a = aX_1^4 \\
F &= B+E = 3X_1^2+aX_1^4 \\
G &= F = 3X_1^2+aX_1^4)^2 \\
H &= Y_1^2 \\
I &= X_1H = X_1 Y_1^2 \\
J &= I.8= 8Y_1^2X_1 \\
K &= G - J = (3X_1^2+aX_1^4)^2 - 8Y_1^2X_1=X_4 \\
K &= 4I= 4X_1Y_1^2 \\
L &= K - X_3=4X_1Y_1^2 - X_3 \\
M &= H^2 = Y_1^4 \\
N &= 8M = 8Y_1^4 \\
\end{align*}
\]
\[ O = FL = (3X_1^2 + aX_1^4)(4X_1Y_1^2 - X_3) \]
\[ P = N - O = (3X_1^2 + aX_1^4)(4X_1Y_1^2 - X_3) - 8Y_1^4 = Y_4 \]
\[ Q = Y_1Z_1 \]
\[ R = 2Q = 2Y_1Z_1 = Z_4 \]

Here, the point doubling is measured based on the sum of 6 point squaring, 3 point multiplication, 6 point multiplication with constant value, 3 point subtraction, 1 point addition without point inversion of 2P. The following parameters are considered to evaluate the execution time of this given formula:

- **M**: the cost of a field multiplication
- **S**: the cost of a field squaring
- **Ma**: the cost of the multiplications of fixed parameters such as the curve parameters a and b

The additions and subtractions are assumed to have the same time complexity of affine point operations. Here, the computational cost of a point addition is redefined as 7M + 4S if \( Z_2 = 1 \) and point doubling for 4M + 4S or 1M + 8S + 1Ma.

### 2.6.2 Limitation

The main advantage of this methodology is to remove point inversion in the point multiplication. But it has a lot of data dependencies within projective co-ordinates (Ekambaram Kesavulu Reddy 2011). So it is not suitable for parallel computation. It is only suitable for special type of elliptic curve which may be implemented by using any of the above mentioned point multiplication methodology (Peter L Montgomery 2005). The second demerit is time consuming process for conversion. Each time
affine coordinated system is converted into projective coordinate system and performs point multiplication. Then, it is again converted into affine coordinate system.

The survey in literature shows that there is no optimal and proper implementation of the point multiplication for parallel processing. So the code scheduling concept is suggested for optimizing point multiplication of EC over prime or binary field through parallel processing. This innovative technique is implemented with help of the divide and conquer strategy, which breaks the point multiplication into two or more sub-problems, until these become simple to compute directly with the help of the point addition or doubling. Finally, these solutions are combined together to solve point multiplication.