SM2 的倍点运算快速实现

背景：

椭圆曲线密码算法的快速实现一直是椭圆曲线密码体制研究的重点。原因在于椭圆曲线的相关常规运算速度较慢，相较于对称密码算法的实现效率很低。而SM2算法作为由国家密码管理部门批准的基于椭圆曲线的非对称密码算法，在我国密码行业中被广泛使用，因此如何优化SM2中的结构，使得SM2算法在实际应用中拥有较高的实现效率成了亟待解决的问题。

基于SM2倍点运算的快速实现，即实现如下操作的快速运算：

$$对于位于椭圆曲线上的点P及整数k，计算[k]P$$

分析：

为加速椭圆曲线上的倍点运算，一个很好的思路是基于点加运算，利用类似于取模运算中的快速幂方法有效的减少点加运算进而实现快速运算。由于该方法需要用到k的二进制表示方式，因此在这里我们将该方法记为二进制表示法。但通过编程实现分析，相较于目前主流的一些密码学库在速度方面还是存在较大差异。综和上面的情况，对椭圆曲线上的倍点快速实现主要从以下两个角度着手：

- 优化二进制表示法基于的点加子运算
- 优化二进制表示法，使得计算过程尽可能地减少点加以及倍点运算

优化：

1.基于射影坐标系实现点加运算的快速运算

上网查阅几种坐标系下的椭圆曲线的相关运算后，综合比较运算复杂度后，选择$Jacobi$坐标系实现素域ECC上的点加运算。

对于F_p上的椭圆曲线，当素数p>3时，椭圆曲线在$Jacobian$加重坐标系下的方程为：

$$y^2 = x^3 + axz^4 + bz^6$$

其中a,b仍须满足：

$$4a^3 + 27b^2在模p意义下不为0$$

在$Jacobian$加重坐标系下，椭圆曲线上的点按如下运算规则构成加法交换群：

1. $O + O = O$
2. $\forall P = (x, y, z) \in E(F_q)\backslash\{O\}$，$P + O = P$
3. $\forall P = (x, y, z) \in E(F_q)\backslash\{O\}$，$P$的负元$-P = (x, p - y, z)$，且$P + (-P) = O$
4. 设$P_1 = (x_1, y_1, z_1) \in E(F_q)\backslash\{O\}$，$P_2 = (x_2, y_2, z_2) \in E(F_q)\backslash\{O\}$
   $P_3 = (x_3, y_3, z_3) = P_1 + P_2 \neq O$，有如下计算方法：

（详见链接：http://hyperelliptic.org/EFD/g1p/auto-shortw-jacobian.html#addition-add-2007-bl）

这里选用最快的加法公式和倍点公式：

- **加法add：**
  - The "add-2007-bl" addition formulas [database entry; Sage verification script; Sage output; three-operand code]:
    - Cost: 11M + 5S + 9add + 4*2.
    - Cost: 10M + 4S + 9add + 4*2 dependent upon the first point.
    - Source: 2007 Bernstein–Lange; note that the improvement from 12M+4S to 11M+5S was already mentioned in 2001 Bernstein http://cr.yp.to/talks.html#2001.10.29.
    - Explicit formulas:

$$
\begin{aligned}
Z1Z1 &= Z_1^2 \\
Z2Z2 &= Z_2^2 \\
U1 &= X_1 * Z2Z2 \\
U2 &= X_2 * Z1Z1 \\
S1 &= Y_1 * Z_2 * Z2Z2 \\
S2 &= Y_2 * Z_1 * Z1Z1 \\
H &= U2 - U1 \\
I &= (2 * H)^2 \\
J &= H * I \\
r &= 2 * (S2 - S1) \\
V &= U1 * I \\
X_3 &= r^2 - J - 2 * V \\
Y_3 &= r * (V - X_3) - 2 * S1 * J \\
Z_3 &= ((Z_1 + Z_2)^2 - Z1Z1 - Z2Z2) * H
\end{aligned}
$$

  - The "mmadd-2007-bl" addition formulas [database entry; Sage verification script; Sage output; three-operand code]:
    - Assumptions:$Z_1 = 1 \, and \, Z_2 = 1$.
    - Cost: 4M + 2S + 6add + 4*2 + 1*4.
    - Source: 2007 Bernstein–Lange.
    - Explicit formulas:

$$
\begin{aligned}
H &= X_2 - X_1 \\
HH &= H^2 \\
I &= 4 * HH \\
J &= H * I \\
r &= 2 * (Y_2 - Y_1) \\
V &= X_1 * I \\
X_3 &= r^2 - J - 2 * V \\
Y_3 &= r * (V - X_3) - 2 * Y_1 * J \\
Z_3 &= 2 * H
\end{aligned}
$$

- **倍点double:**
  - The "dbl-2007-bl" doubling formulas [database entry; Sage verification script; Sage output; three-operand code]:
    - Cost: 1M + 8S + 1*a + 10add + 2*2 + 1*3 + 1*8.
    - Source: 2007 Bernstein–Lange.
    - Explicit formulas:

$$
\begin{aligned}
XX &= X_1^2 \\
YY &= Y_1^2 \\
YYYY &= YY^2 \\
ZZ &= Z_1^2 \\
S &= 2 * ((X_1 + YY)^2 - XX - YYYY) \\
M &= 3 * XX + a * ZZ^2 \\
T &= M^2 - 2 * S \\
X_3 &= T \\
Y_3 &= M * (S - T) - 8 * YYYY \\
Z_3 &= (Y_1 + Z_1)2 - YY - ZZ
\end{aligned}
$$

  - The "mdbl-2007-bl" doubling formulas [database entry; Sage verification script; Sage output; three-operand code]:
    - Assumptions: $Z_1 = 1$.
    - Cost: 1M + 5S + 7add + 3*2 + 1*3 + 1*8.
    - Source: 2007 Bernstein–Lange.
    - Explicit formulas:

$$
\begin{aligned}
XX &= X1^2 \\
YY &= Y1^2 \\
YYYY &= YY^2 \\
S &= 2 * ((X_1 + YY)^2 - XX - YYYY) \\
M &= 3 * XX + a \\
T &= M^2 - 2 * S \\
X_3 &= T \\
Y_3 &= M * (S - T) - 8 * YYYY \\
Z_3 &= 2 * Y_1
\end{aligned}
$$

2.基于NAF优化二进制表示法

在已有类似于快速幂的乘化加的椭圆曲线倍点运算上，可以继续使用NAF表示法减少点加运算，进而缩短运算时间，提高运算效率。

- **引理1：NAF（非邻接）二进制表示的正整数的非零位平均密度为1/3。**

  证明：

  记a_n表示n位NAF数字的个数，则有$a_1 = 1, a_2 = 1, a_3 = 3$。考虑$a_n$的最低位：

  1. 最低位是0的n位NAF数有a_{n-1}个，
  2. 最低位为1或-1的n位NAF数分别有a_{n-2}个

  因此：$a_n = a_{n-1} + 2a_{n-2}$。

  记v_n表示所有n位NAF 数字非零位个数的总和，则有$v_1 = 1, v_2 = 1, v_3 = 5$。考虑$v_n$的最低位:

  1. 最低位是0的n位NAF数的非零位个数之和等于v_{n-1}
  2. 最低位为1的n位NAF数有a_{n-2}个，若不考虑最低位则这样的数的非零位个数之和为v_{n-2}，再算上最低位的a_{n-2} 个数字，每个1位。总计最低位是1的n位NAF数的非零位个数之和为 $v_{n-2} + a_{n-2}$
  3. 最低位为-1同理可得

  因此：$v_n = v_{n-1} + 2v_{n-2} + 2a_{n-2}$。

  于是有：

  $$v_n = v_{n-1} + 2v_{n-2} + 2a_{n-2}$$
  $$v_{n-1} = v_{n-2} + 2v_{n-3} + 2a_{n-3}$$
  $$v_{n-2} = v_{n-3} + 2v_{n-4} + 2a_{n-4}$$

  处理上面的三个式子便可得出：

  $$v_n = 2v_{n-1} + 3v_{n-2} - 4v_{n-3} - 4v_{n-4}$$

  接下来对a_n与v_n用特征根法求通项，就能得到：

  $$a_n = (-\frac{1}{3})(-1)^n + (\frac{1}{3})2^n$$
  $$v_n = \frac{6n+10}{27}2^{n-1} + \frac{6n+5}{27}(-1)^{n-1}$$

  最后可求得：

  $$lim\frac{v_n}{na_n} = \frac{1}{3}$$

  注：上述求得的非零位平均密度又叫做汉明密度

- **引理2：标准二进制表示的正整数的非零位平均密度为1/2。**

  证明略；

- **NAF与标准快速运算的操作比较：**

比特长度为l的K	标准快速运算	NAF
点加运算	l/2-1	l/3-1
倍点运算	l-1	l-1

- **伪代码：**

Algorithm 1 $NAF(k)$

Input: k
Output: $NAF(k)$

while $k > 1$ **do**
  if $k\%2 == 1$ **then**
    $temp = 2 - k\%4$
    $NAF[].append(temp)$
    $k = k - temp$
  else
    $NAF[].append(0)$
  end if
  $k = k//2$
end while
return NAF[]

- **w-NAF**

由于在仿射坐标系以及Jacobian坐标系上的点加减运算实现效率上相差不大，因此我们可以对上面的NAF表示再做处理，使得二进制表示方式中非零比特的数量进一步缩小，从而提高多倍点的运算速度。

通过类比上面的普通NAF表示，我们可归纳出这样的拓展NAF表示：

1. 当$w = 2$时，$NAF_2(K)$就是k的非邻近表示形式$NAF(K)$
2. **对于一个比特长度为l的数k：存在唯一的$NAF_w(k)$表示形式，其汉明密度为：** $\frac{l}{w+1}$

- **伪代码：**

Algorithm 1 $NAF_w(k)$

Input: k
Output: $NAF_w(k)$

$i = 0$
while $k \geq 1$ **do**
  if $k\%2 == 1$ **then**
    $k_i = k \bmod 2^w$
    if $k_i > 2^{w-1} - 1$ **then**
      $k_i = k_i - 2^w$
    end if
    $k = k - k_i$
  else
    $k_i = 0$
  end if
  $k = k//2$
  $i = i + 1$
end while
return $(k_{l-1}, k_{l-2}, ..., k_1, k_0)$

3.基于滑动窗口预计算加速二进制表示法

在计算得k的$NAF_w(k)$表示后，我们进一步考虑优化加速问题：在常用的二进制展开法中，由于二进制位零与非零的分布不均匀，导致每次计算时只能一个二进制位一个二进制位迭代计算。但有了NAF_w表示后，二进制表示中不会出现连续的非零位，这样的话我们就能以w个二进制位为一个整体，先预计算出$1P$至$(2^w - 1)P$的值，相当于2^w进制下的展开方式加速。

- **基于二进制展开的滑动窗口法伪代码：**

Algorithm 1 基于二进制展开的滑动窗口算法

Input: P，整数k，NAF(k)为长度为l的01比特串，窗口大小w
Output: Q= [k]P

$P_1 = P, P_2 = [2]P$
i 从1至$2^{w-1} - 1$ 计算: $P_{2i+1} = P_{2i-1} + P_2$
令j = l-1， Q = O
while $j \geq 0$ **do**
  if $k_j = 0$ **then**
    $Q = [2]Q$
    $j = j - 1$
  else
    令t是使$j - t + 1 \leq w$且$k_t = 1$的最小非负整数
    $h_j = k_t 2^0 + k_{t+1} 2^1 + ... + k_j 2^{j-t}$
    $Q = [2^{j-t+1}]Q + P_{h_j}$
    j = t-1
  end if
end while
return Q

- 基于NAF_w展开的滑动窗口法伪代码与上面类似，在这里不再赘述
- 详见：SM2椭圆曲线公钥密码算法的快速实现研究

4.优化结果分析：

通过编写测试程序，与标准库函数进行比较，得到如下的结果：

测试样例如下：

```
椭圆曲线参数中的大素
数:115792089210356248756420345214020892766250353991924191454421193933289684991999
椭圆曲线参数
a:115792089210356248756420345214020892766250353991924191454421193933289684991996
椭圆曲线参数
b:18505919022281880113072981827955639221458448578012075254857346196103069175443
点P坐标
x:304754800878696144434700891277069833866759377062631212236195948394055352691 40

点P坐标
y:900236012396435171589397905109089517070018767734879507102524048546092123606 54
k:617136796639221155023273542791931252086147246828731258960128457053519788
```

测试结果如下：

```
"E:\E_drive\clion\Project List\cryptography\Crypto_Experiment\cmake-build-debug\Exp11--test.exe"
115792089210356248756420345214020892766250353991924191454421193933289684991999
115792089210356248756420345214020892766250353991924191454421193933289684991996
185059190222818801130729818279556392214584485780120752548573461961030691754 43
304754800878696144434700891277069833866759377062631212236195948394055352691 40 900236012396435171589397905109089517070018767734879507102524048546092123606 54
617136796639221155023273542791931252086147246828731258960128457053519788
slid_window_smul speed:1.677852/s
standard_smul speed:2.631579/s
878218229949103565540736652221324560833965461221793255230242114227084668199 4 781444684286484517204996585671741588978034035069954615468322586230347562593 83

Process finished with exit code 0
```

限于时间原因，所作的优化并不完善，一是滑动窗口算法在细节上没有完全优化，对于点加坐标系的选取还有更好的选择比如Lopez&Dahab坐标系下的效率更高，因而在速度上仍与标准库实现存在差距。在今后的学习与研究中还会继续优化完善。

5.参考文献：

1. http://hyperelliptic.org/EFD/g1p/auto-shortw-jacobian.html#addition-add-2007-bl
2. SM2椭圆曲线公钥密码算法的快速实现研究，牛永川，王小云。山东大学2013
3. SM2算法快速实现，孙宏建。电子科技大学2021
4. Bernstein-Lange2007_Chapter_FasterAdditionAndDoublingOnEll
5. Renes2016_Chapter_CompleteAdditionFormulasForPri
6. 用于椭圆曲线上的点的通用计算方法