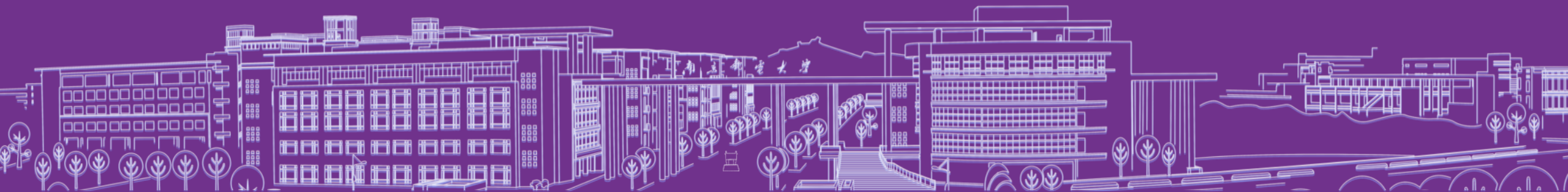
南京邮电大学
Nanjing University of Posts and Telecommunications

基于浮点数计算能力的高性能椭圆曲线密码实现技术研究

DPF-ECC: Accelerating Elliptic Curve Cryptography with Floating-Point Computing Power of GPUs

汇报人：吴 雯　　指导老师：董建阔



南京郵電大學 Nanjing University of Posts and Telecommunications

目录

CONTENT

研究背景

海量的应用场景+用户需求

↓

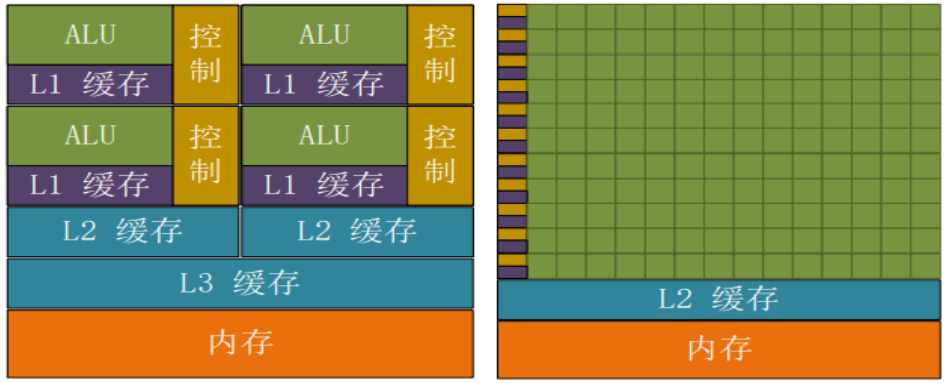
高吞吐量、低延时

↓

加速基于大数运算的公钥密码算法

↓

研究基于CPU、FPGA、ASIC、GPU等平台的**高速**实现



GPU：数据处理➔并行计算

➔高指令吞吐量、高内存带宽

椭圆曲线密码(ECC)可以使用更短的密钥提供与 RSA 密码系统相同安全级别的功能，其广泛应用于数字签名、密钥交换协议等

RSA 密钥大小 (bit)	ECC密钥大小 (bit)
1024	160
2048	224
3072	256
7680	384
15360	521

Curve/Edwards25519的密钥大小为 256 比特，是一个定义在素数域 $\mathbb{F}p$（$p = 2^{255} - 19$），具有 128 比特安全性的曲线

Curve25519 蒙哥马利形式的曲线方程：

$$y^2 = x^3 + 486662\,x^2 + x.$$

Edwards25519扭曲的爱德华兹曲线方程：

$$ax^2 + y^2 = 1 + dx^2y^2$$

其中，**模乘**运算是主要的运算负载➔**优化**

目录

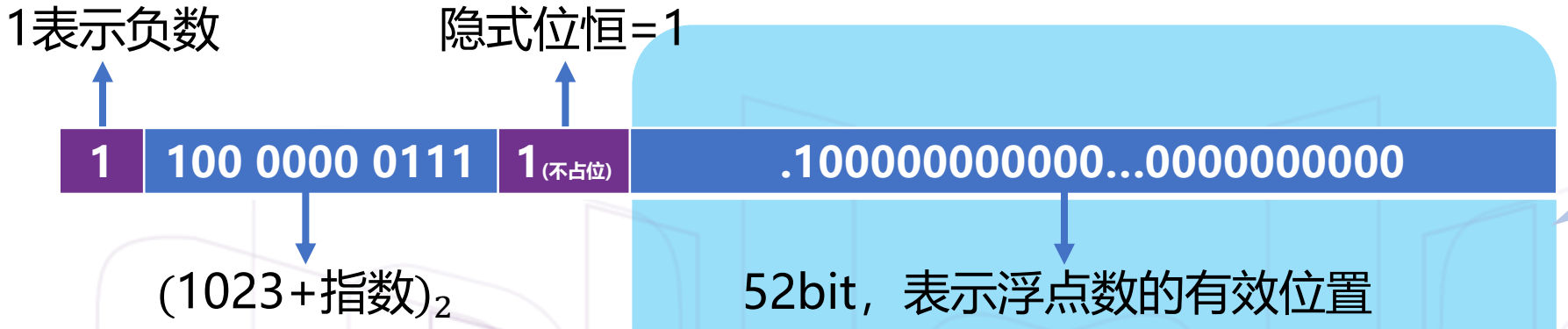
CONTENT

预备知识：double数据类型

双精度浮点数的格式：

符号位(1bit)	指数位(11bit)	隐式位(不占位)	尾数位(52bit)

以$(-384)_{10}$为例，$(384)_{10}$的二进制表示为$(1\ 1000\ 0000)_2$，
从而$(-384)_{10}$可以表示为 $(-1)^1 \times 2^8 \times (1.1)_2$，其浮点数格式如下：



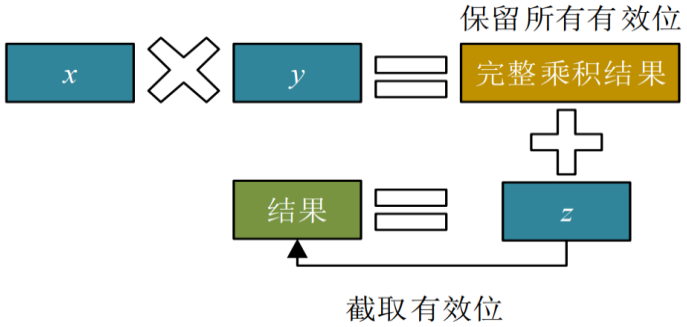
提出方法：利用双精度浮点数的**52bit尾数**部分存储大整数！

注：不是所有的数都能被精确表示，如用double类型表示2/3就会丢失精度
IEEE754标准规定了4种规约方式，本研究采用的是round towards zero(rz)

预备知识：积和熔加FMA

对于乘加运算：$(x \times y) + z$

1. 普通乘加运算分别在乘法和加法计算结束后各进行一次rz舍入操作，由此会带来更多的精度损失
2. 积和熔加方法(**FMA**)仅在乘法和加法都计算完成后进行rz舍入操作，因此操作**更快**、结果**更准确**



FMA计算机制

(1) $A = 1 \times (1.00000000\ 00000000\ 0000001)_2$
(2) $B = -1 \times (1.00000000\ 00000000\ 0000001)_2$
(3) $rz(A \times A + B) = 2^{-46}$ [积和熔加计算结果]
(4) $rz(rz(A \times A) + B) = 0$ [普通乘加计算结果]

积和熔加与常规乘加的区别

Dekker单字乘法

输入: double类型的乘数 a 与被乘数 b
输出: 低位子积 c_lo 和高位子积 c_hi
(1) $c\_hi = \_fma\_rz(a, b, 0.0)$
(2) $c\_lo = \_fma\_rz(a, b, -c\_hi)$

步骤(1)使得$a \cdot b$ 的结果仅保留高 53 位有效数字(整数位和分数部分)

步骤(2)用完整乘积$-c_hi$，得到低位子积，但c_lo 的位数不固定

子积精度不同，累加过程无法对齐

如何改进?

目录

CONTENT

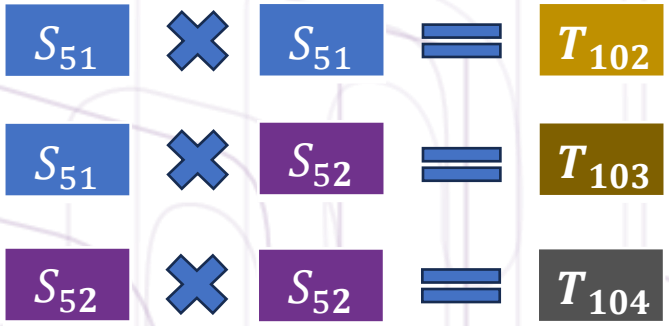
03

有限域运算：大数表示

Curve/Edwards25519的计算基于255位素数域上的$F_p(p = 2^{255} - 19)$
本研究使用4个51位的字和1个52位的字**冗余**表示GF$(2^{255} - 19)$ 上的大整数：

S_{51}：使用1个double表示、字长为 51的字
S_{52}：使用1个double表示、字长为 52的字
S_{256}：使用5个double表示的256位大整数，其中前4个double为S_{51}，最后1个double为S_{52}

乘法过程中出现三种单字乘法的情形：



T_{102}（T_{103}、T_{104}）往往无法存储在单个double类型的尾数部分，Dekker方法会导致累加过程无法对齐

Dekker方法的改进：(更灵活、更高效)

输入: S_{51}或S_{52} 类型表示的a 与 b
输出: 低位子积 c_lo 和高位子积 c_hi
(1) $p\_hi = \_fma\_rz(a, b, 2^{103})$
(2) $p\_lo = \_fma\_rz(a, b, 2^{103} + 2^{52} - p\_hi)$

对低位子积的位数有**锚定**作用，
可**直接**用于后续多精度乘法的累加计算

有限域运算：高/低位子积

T_{102}和T_{103}的高位子积：

$a_i \times b_j$的乘积结果$< 2^{103}$，$p\_hi = \_fma\_rz(a, b, 2^{103})$ 中2^{103}占主导地位，所以指数位恒为$(1023+103)_{10} = (100\ 0110\ 0110)_2 = (0x466)_{16}$

0	100 0110 0110	1(不占位)	.*****...***

前12bit恒定不变

T_{102}：1bit的0+51bit的高位积
T_{103}：52bit的高位积

T_{102}和T_{103}的低位子积：

$p_lo = _fma_rz(a, b, 2^{103} + 2^{52} - p_hi)$中$2^{52}$占主导地位，所以指数位恒为$(1023+52)_{10} = (100\ 0011\ 0011)_2 = (0x433)_{16}$

0	100 0011 0011	1(不占位)	.0****...***

前12bit恒定不变

T_{102}：1bit的0+51bit的低位积
T_{103}：1bit的0+51bit的低位积

锚定！！

$S_{52} \times S_{52} = T_{104}$

T_{104}的高/低位子积：

乘法结果为T_{104}类型的最长为104bit，理论上是拆分成2个52bit的高/低位积，用同样的公式还能锚定低位积的位数吗？

后面证明：T_{104}类型可以用同样的公式计算，其子积的表示形式**与T_{103}类型相同**

有限域运算：掩码操作

在计算乘积结果时涉及到累加过程， 若不对double类型的数据进行**掩码**处理，累加过程中可能会出现溢出，从而导致运算错误

1. 将存有高/低位子积的双精度浮点数直接转化为**uint64_t**格式的二进制，并在**整数域**中进行求和

0100	0110	0110	(52bit，存储高位子积)
0100	0011	0011	(52bit，存储低位子积)

1: **for** i = 0 to 4 **do**
2: $c_i = i \times 0x466 + (i + 1) \times 0x433$
3: $c_i = -(c_i \ \& \ 0xFFF) << 52$
4: $c_{9-i} = (i + 1) \times 0x466 + i \times 0x433$
5: $c_{9-i} = -(c_{9-i} \ \& \ 0xFFF) << 52$
6: **end for**

2. 累加器中生成负初始值，以抵消浮点数的符号位和幂指数位

0100	0110	0110	(52bit，存储高位子积)

前12bit恒定不变

T_{102}：1bit的0+51bit的高位积
T_{103}、T_{104}：52bit的高位积

0100	0011	0011	(52bit，存储低位子积)

前12bit恒定不变

T_{102}、T_{103}、T_{104}：
1bit的0+51bit的低位积

0000	0000	0000	(52bit，存储高位子积)

T_{102}：1bit的0+51bit的高位积
T_{103}、T_{104}：52bit的高位积

0000	0000	0000	(52bit，存储低位子积)

T_{102}、T_{103}、T_{104}：
1bit的0+51bit的低位积

此时每一列累加器最多可以累加2^{11}项子积，而这里至多累加10项，不会溢出

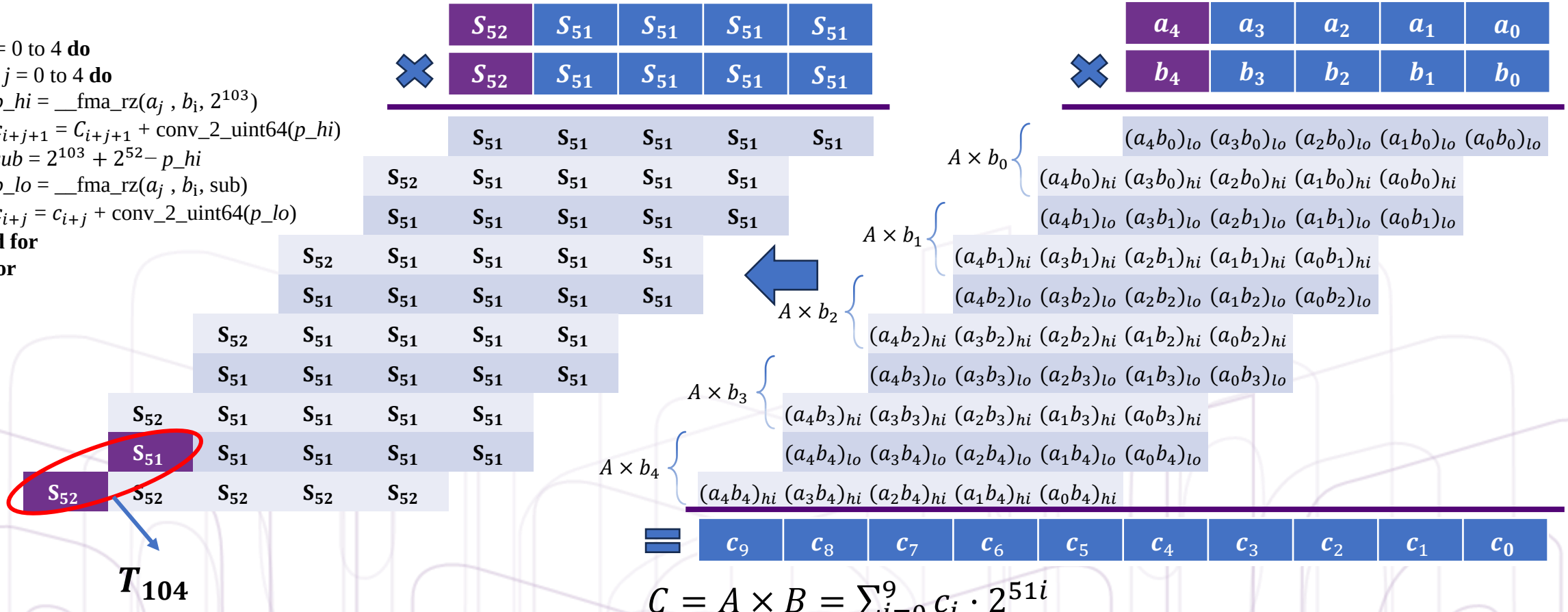
有限域运算：多精度乘法

将两个GF($2^{255}-19$) 上的大整数 A 和 B 表示成S_{256} 格式：

$A=\sum_{i=0}^{4} a_i \cdot 2^{51i}$，$B=\sum_{i=0}^{4} b_{\mathrm{i}} \cdot 2^{51i}$，当 $0 \le i < 4$ 时，$0 \le \mathrm{a_i}, b_{\mathrm{i}} < 2^{51}$，当 $i=4$ 时，$0 \le \mathrm{a_i}, b_{\mathrm{i}} < 2^{52}$

乘法$A \times B$可以拆分成若个单精度乘法$a_i \times b_j$，$0 \le i, \mathrm{j} \le 4$

1: **for** i = 0 to 4 **do**
2: **for** j = 0 to 4 **do**
3: p_hi = __fma_rz(a_j , b_{i}, 2^{103})
4: $c_{i+j+1} = C_{i+j+1}$ + conv_2_uint64(p_hi)
5: $sub = 2^{103} + 2^{52} - p\_hi$
6: p_lo = __fma_rz(a_j , b_{i}, sub)
7: $c_{i+j} = c_{i+j}$ + conv_2_uint64(p_lo)
8: **end for**
9: **end for**



$$C = A \times B = \sum_{i=0}^{9} c_i \cdot 2^{51i}$$

有限域运算：快速约减

多精度乘法完成后，乘法结果可用10个冗余的uint64_t类型的累加器表示为$C = \sum_{i=0}^{9} c_i \cdot 2^{51i}$

$p = 2^{255} - 19 \;\rightarrow\; 2^{255} \equiv 19 \; mod \; p$

快速约减公式：

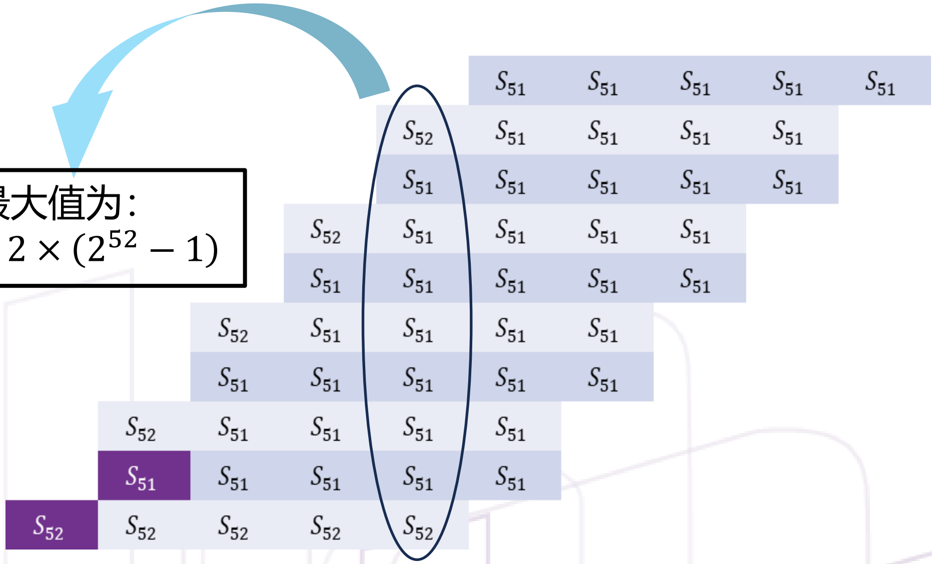
$$
\begin{aligned}
C &= \sum_{i=0}^{9} c_i \cdot 2^{51i} \\
&= \sum_{i=0}^{4} c_i \cdot 2^{51i} + \sum_{i=5}^{9} c_i \cdot 2^{51i} \\
&= \sum_{i=0}^{4} c_i \cdot 2^{51i} + \sum_{i=0}^{4} c_{i+5} \cdot 2^{51(i+5)} \\
&= \sum_{i=0}^{4} c_i \cdot 2^{51i} + 2^{255} \cdot \sum_{i=0}^{4} c_{i+5} \cdot 2^{51i} \\
&\equiv \sum_{i=0}^{4} c_i \cdot 2^{51i} + \sum_{i=0}^{4} 19 \cdot c_{i+5} \cdot 2^{51i} \; (mod \; p) \\
&\equiv \sum_{i=0}^{4} (c_i + 19 \cdot c_{i+5}) \cdot 2^{51i} \; (mod \; p)
\end{aligned}
$$

目的：将 512 位累加器约减至 **256** 位

每个uint64_t累加器：..**(长64bit，最多占59bit)**

通常在快速约减之前要将10个冗余的uint64_t转换成简化形式，本方案采取的51bit字表示可以**跳过“简化”**步骤

每个累加器的最大值为：
$7 \times (2^{51} - 1) + 2 \times (2^{52} - 1)$



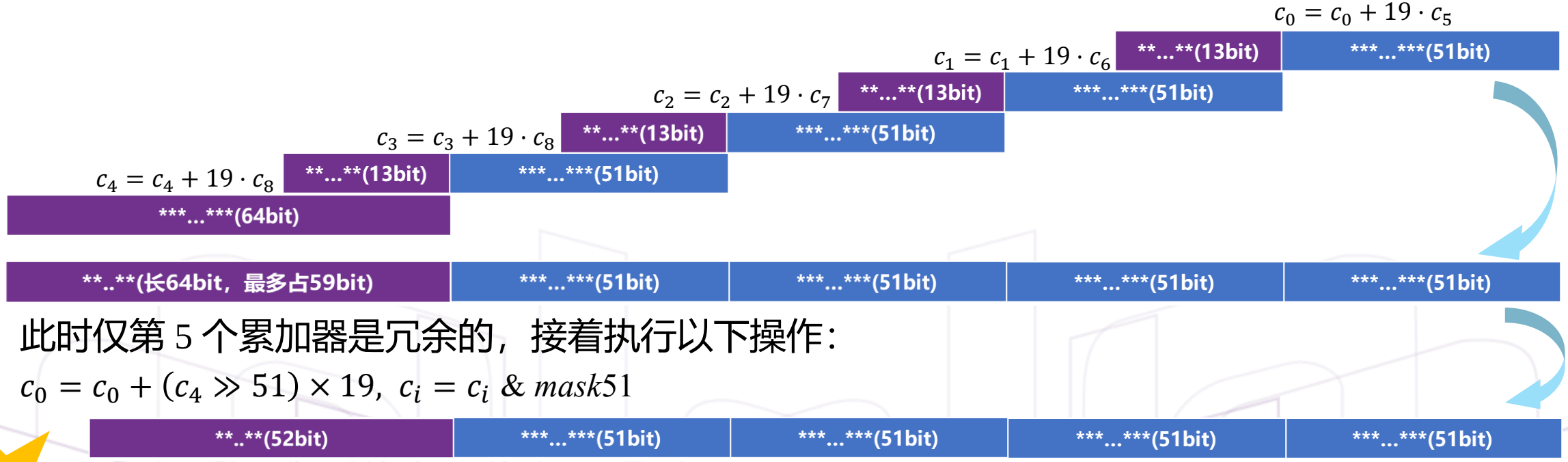
快速约减中，19倍的c_{i+5}被累加至c_i， c_i的最大值为：
$19 \times [7 \times (2^{51} - 1) + 2 \times (2^{52} - 1)] + (2^{51} - 1) < 2^{59} - 1$
不会超过累加器(64bit)的取值范围

有限域运算：进位消解

快速约减后得到5个冗余的uint64_t类型的数值，而不是精确的S_{256} 格式

对于前 4 个累加器，将每一个uint64_t数值的高 13 位作为进位累加至下一个uint64_t数值

$c_{i+1} = c_{i+1} + (c_i \gg 51),\ c_i = c_i\ \&\ mask51,\ i \epsilon [0,3], mask51 = 0x7FFFFFFFFFFFFull$



此时仅第 5 个累加器是冗余的，接着执行以下操作：

$c_0 = c_0 + (c_4 \gg 51) \times 19,\ c_i = c_i\ \&\ mask51$

经过一轮**进位消解**(执行一次以上两个步骤)，5 个累加器的结果限定在$[0, 2^{256})$，因为

$$19 \times (2^{59-51} - 1) + 2^{255} - 1 < 2^{255} + 4845 < 2^{256} - 1$$

优点：**无额外开销**的同时，可以**直接**作为下一个函数的输入

有限域运算： T_{104}高/低位子积

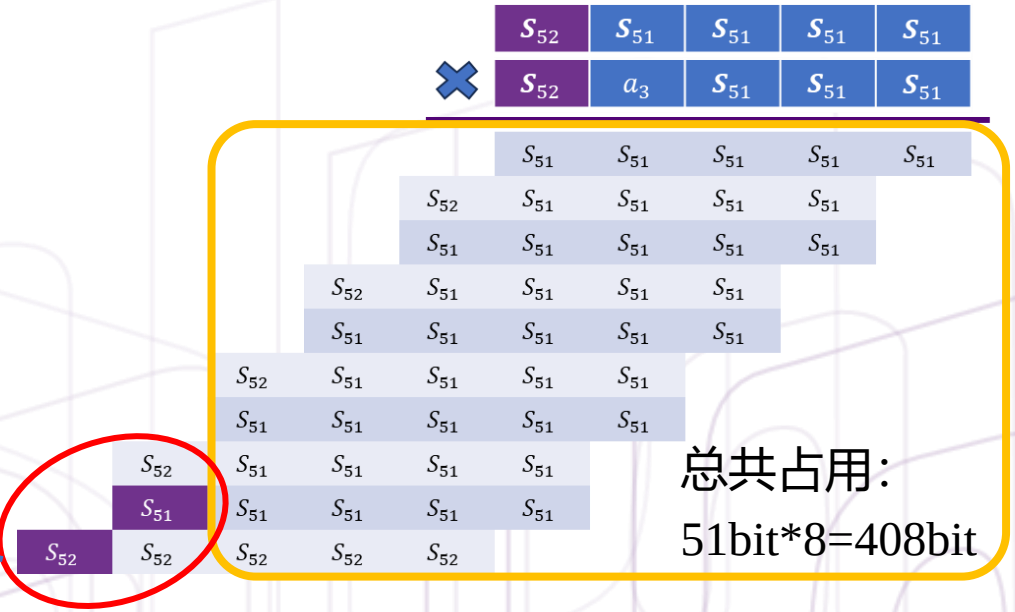
证明：
T_{104}类型的子积的表示形式**与T_{103}类型相同**

①初始时：
本方案中用S_{256} 表示一个256bit的数：

S_{52}	S_{51}	S_{51}	S_{51}	S_{51}

因为大数运算基于GF($2^{255}-19$)，初始时A和B的最高位字实际仅51bit，此时T_{104}不会超过103bit，可以表示成T_{103}类型的格式；

②运算过程中：
乘法过程中，乘法结果的取值范围($< 2^{255}+4845$)，
因为大整数的取值范围足够小，所以在进行乘法运算时，
两个大整数的乘法结果 ($< (2^{255}+4845)^2$) 比 2^{551} 小，
最高两个字的乘积**不会超过103bit**



目录

CONTENT

04

点算术运算优化： 点加 倍点

拓展坐标：(U,V,Z,T) ⟷ $x=U/Z, y=V/Z, xy=T/Z$ ⟷ 仿射坐标：(x,y)

点加：

$P_2(U_2,V_2,Z_2,T_2)=P_1(U_1,V_1,Z_1,T_1)+P_2(U_2,V_2,Z_2,T_2)$

$(1)Z_2=Z_2\times Z_1$ $(2)Z_1=Z_2+2$ $(3)Z_2=T_1\times T_2$

$(4)T_2=d_2\times Z_2$ $(5)Z_2=V_2-U_2$ $(6)T_1=V_2+U_2$

$(7)U_2=V_1-U_1$ $(8)V_2=Z_2\times U_2$ $(9)Z_2=V_1+U_1$

$(10)U_2=T_1\times Z_2$ $(11)U_1=U_2+V_2$ $(12)V_1=Z_1-T_2$

$(13)T_1=Z_1+T_2$ $(14)Z_1=U_2+V_2$ $(15)U_2=U_1\times V_1$

$(16)V_2=T_1\times Z_1$ $(17)Z_2=V_1\times T_1$ $(18)T_2=U_1\times Z_1$

8个变量、9次模乘、6次模加、3次模减

简化算法计算步骤、重用变量、减少对寄存器的使用！

混合点加：寄存器的利用更有优势

$P_2(U_2,V_2,Z_2,T_2)=P_1(X_1,Y_1)+P_2(U_2,V_2,Z_2,T_2)$

因$P_1(X_1,Y_1)=(X_1,Y_1,Z_1=1,T_1=X_1\times Y_1)$

故移除步骤(1)，同时添加$T_1=X_1\times Y_1$

6个变量、9次模乘、6次模加、3次模减

倍点：执行步骤少

$P_1=[2]P_1$，即$P_1=P_1+P_1$，$P_1=(U_1,V_1,Z_1,T_1)$

$(1)M=U_1^2$ $(2)N=V_1^2$ $(3)T_1=U_1+V_1$ $(4)U_1=M+N$

$(5)V_1=M-N$ $(6)M=T_1^2$ $(7)N=U_1-M$ $(8)T_1=Z_1^2$

$(9)M=T_1+V_1$ $(10)Z_1=M\times V_1$ $(11)T_1=N\times U_1$

$(12)V_1=V_1\times U_1$ $(13)U_1=M\times N$

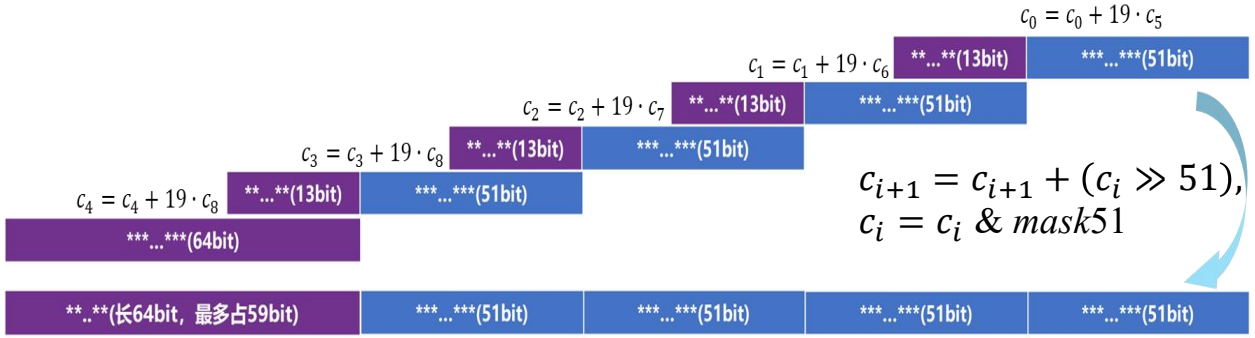
6个变量、4次模乘、4次模平方、3次模加、2次模减

点算术运算优化： 点乘

简化算法计算步骤、重用变量、减少对寄存器的使用！

Curve25519的点乘

若出现：$z_1 = x_1 \times y_1$ ， $z_2 = x_2 \times y_2$ ， $z_2 = z_1 + z_2$ ，
只需要将两个512bit的乘积转换为263bit



Edwards25519的**已知点**点乘

分段算法：访存次数少、点算术运算步骤少、需要大量的存储资源 → 将长k bit的标量 d 分为 n 组，即 $d = d_{n-1} \mid \ldots \mid d_1 \mid d_0$ ，每组包含$2^{k/n}$个点 → 将这n组包含的2^k个点生成**预计算表**存储在GPU**全局内存** → $$[d]G = \sum_{i=0}^{n-1} d_i \cdot 2^{i \cdot k/n} G$$

Edwards25519的**未知点**点乘

固定窗口算法：效率介于Double and add 算法和滑动窗口算法之间、可以**抵御计时攻击** → 每次扫描k bit标量d的ωbit ，生成包含2^ω个点的在线预计算表，存储在**全局内存** → $2^\omega - 1$次点加生成预计算表；$(n-1) \cdot \omega$次倍点、 $(n-1)$次点加生成最终的点乘结果

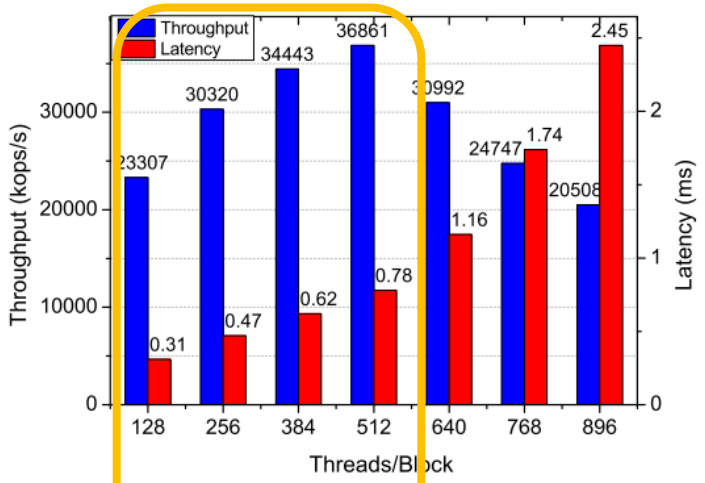
南京郵電大學 Nanjing University of Posts and Telecommunications

目录

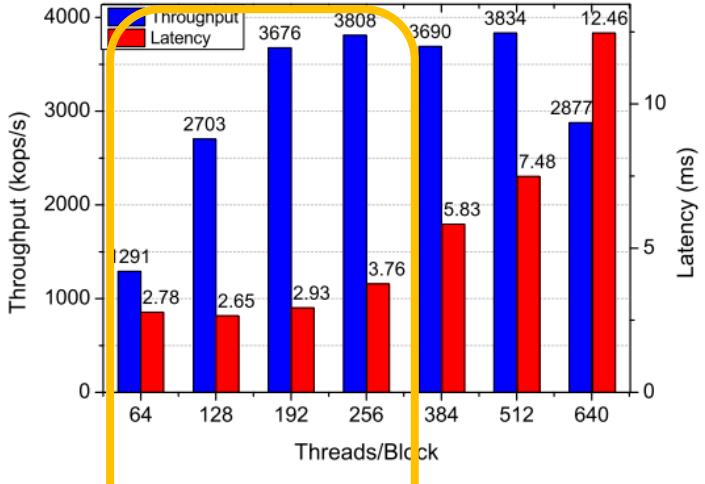
CONTENT

05

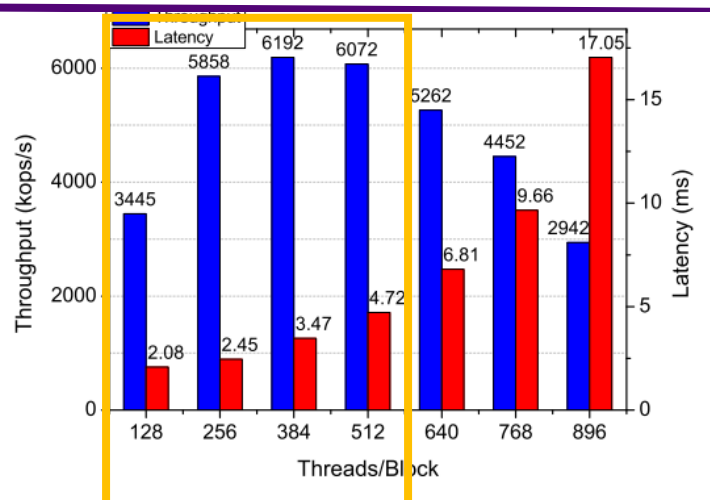
实验分析：不同平台的性能比较



(a) Known Point Multiplication of Edwards25519

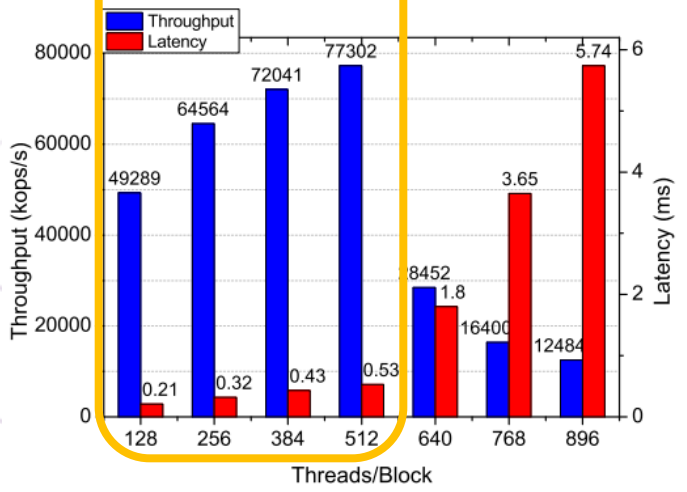


(b) Unknown Point Multiplication of Edwards25519

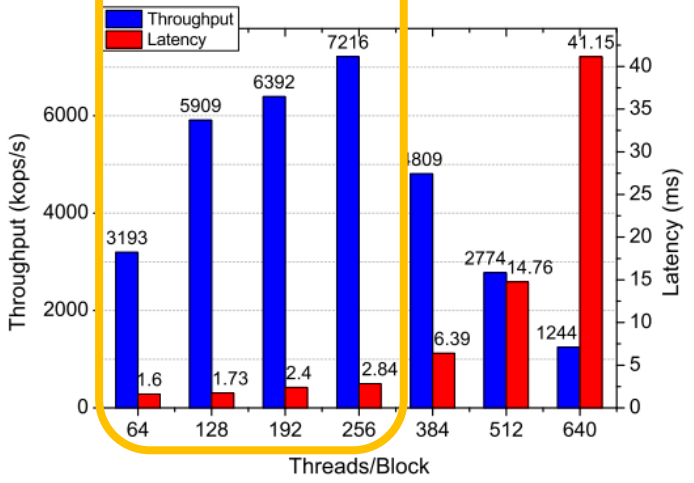


(c) Unknown Point Multiplication of Curve25519

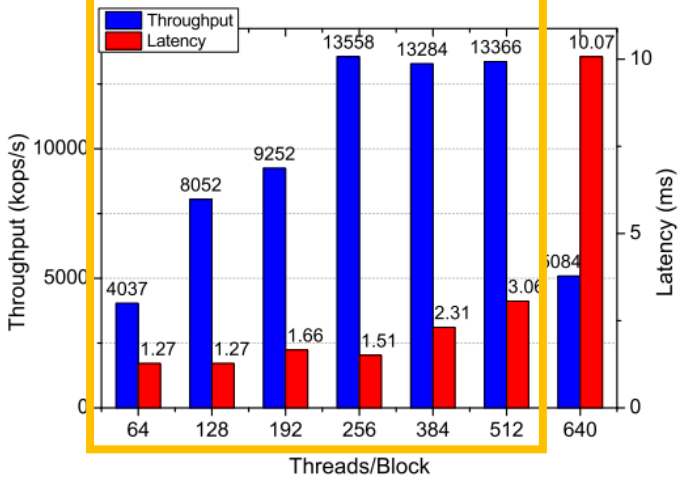
Tesla P100(56个Block)



(a) Known Point Multiplication of Edwards25519



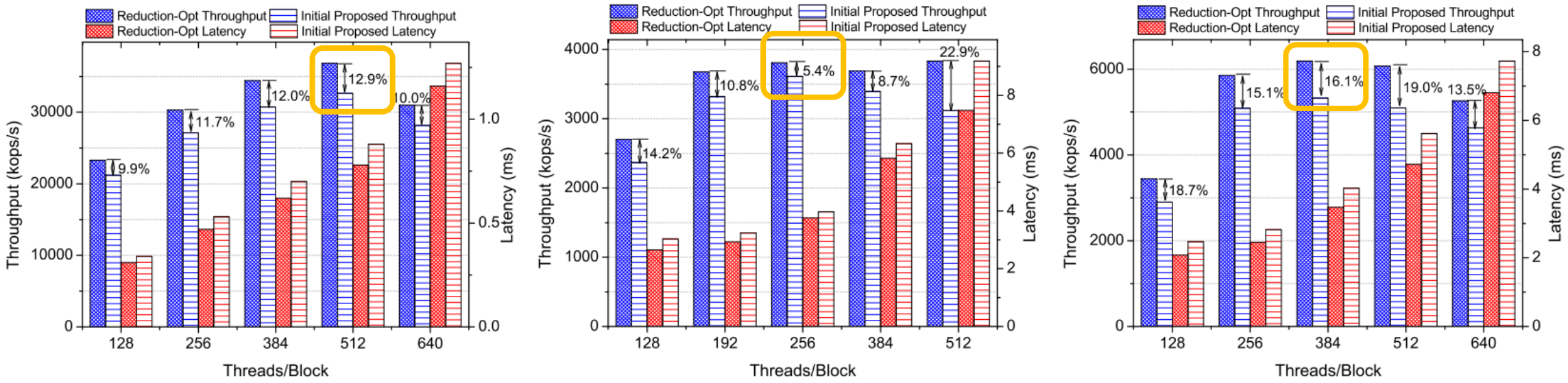
(b) Unknown Point Multiplication of Edwards25519



(c) Unknown Point Multiplication of Curve25519

TITAN V(80个Block)

实验分析：省略“约简”步骤对点乘性能的提升



(a) Known Point Multiplication of Edwards25519 (b) Unknown Point Multiplication of Edwards25519 (c) Unknown Point Multiplication of Curve25519

在快速约减之后跳过繁琐的“约简”步骤也大大提高了性能。如图所示，在 Tesla P100 上，跳过“约简”步骤使得 Edwards25519 的已知和未知点点乘，点乘在峰值处的性能分别**提升了 12.9%, 5.4%, 16.1%**。

实验分析：性能评估

与已有的浮点数实现相比较

	Bernstein et al. [22]	Proposed Method
Coprocessor	GTX 295	TITAN V
GFLOPS	1192.30	14899.2
280-bit Curve by a 11797-bit scalar (ops/s)	400.70	-
Curve25519 (scaled) (Mops/s)	0.552	13.558

即使将平台和曲线的差异考虑在内，本方法的实现也是 Bernstein 等人方法的 **24.56** 倍

与已有的整型数实现相比较

Implementation	GPU	Architecture	GFLOPS[*]	ECC Curve	Known Point (kops)	Unknown Point (kops)	Latency (ms)
Mahe et al. [20]	GTX TITAN	Kepler	4499	Curve25519	-	524	-/-
Dong et al. [21]	GTX TITAN	Kepler	4499	Curve25519	-	1,394	-/-
	GTX 1080	Pascal	7967-8873		-	2,860	-/6.26
	Tesla P100	Pascal	8071-9340		-	3,062	-/7.02
Cui et al. [36]	GTX 285	Tesla	708	224-bit Edwards	345	115	4.52/19.20
Pan et al. [16]	GTX 780Ti	Kepler	5045	NIST P-256	8,710	929	0.42/25.82
Ours	Tesla P100	Pascal	8071-9340	Curve25519	-	**6,192**	-/3.47
	TITAN V	Volta	12288-14899		-	**13,558**	-/1.51
	Tesla P100	Pascal	8071-9340	Edwards25519	**36,861**	**3,808**	0.78/3.76
	TITAN V	Volta	12288-14899		**77,301**	**7,216**	0.53/2.84

主要性能优势在于充分利用了双精度浮点数的有效位、**降低**了快速约减中的**进位消解的轮数**，**省去了“约简”**操作，**合并**了冗余的进位消解步骤，利用了 GPU 的合并访存操作

关于侧信道安全的讨论

本方案中的有限域算术都**不涉及数据依赖型分支**（例如，进位传播等），理论上可以保障程序**免受计时攻击**。点乘中的内存访问和跳转模式也**不依赖标量 k 的任一比特**。因此，无论从曲线层面，还是有限域算术层面，本文的算法实现都**可以抵御计时攻击**。

目录

CONTENT

06

总结推广

- 1. 提出基于双精度浮点数的**大整数乘法计算方法**

将单字乘法的高半部分子积和低半部分子积分别存储在两个双精度浮点数的**尾数部分**，然后通过在 64 位**二进制整数域**对符号位和幂指数位的**掩码操作**消除干扰项，使所得的子积结果无需对齐便可用于多精度列项和的计算。**减少**了大整数乘法计算中的**操作数的数量**和**积和熔加指令的数量**，提升了大整数乘法计算的性能。

- 2. 提出基于双精度浮点数的ECC**点算术运算优化**技术

双精度浮点数的二进制整数域可以容纳更多的冗余位，因此，在完成大整数计算得到列项和之后可以**直接进行快速约减计算，不会出现溢出问题**。这**省略了**在快速约减计算之前将冗余格式转换为标准的大整数表示格式的步骤。本研究将 GF($2^{255}-19$)上的快速约减算法的**进位处理缩减至一轮**，并通过调整算法的执行步骤，合并部分有限域算术间的进位操作提升性能。

- 3. Crandall 素数域($2^k-\sigma$) ➔ Solinas 素数域($2^k-2^t\pm\cdots\pm 1$)

Solinas 素数的形式不如 Crandall 素数的简洁，但是基于 Solinas 素数域的多精度乘法也可以通过**快速约减**算法将乘法结果约减至标准的 DPF 表示格式

南京郵電大學
Nanjing University of Posts and Telecommunications

Thank you for listening