ЧЕБЫШЕВСКИЙ СБОРНИК
Том 13 Выпуск 2 (2012)

Труды IX Международной конференции
Алгебра и теория чисел: современные проблемы и приложения,
посвященной 80-летию профессора Мартина Давидовича
Гриндлингера

АРИФМЕТИКА НА ЭЛЛИПТИЧЕСКИХ КРИВЫХ С ИСПОЛЬЗОВАНИЕМ ГРАФИЧЕСКИХ ВЫЧИСЛИТЕЛЕЙ

П. А. Лебедев, А. Ю. Нестеренко (г. Москва)

Известно, что графические процессоры современных компьютеров могут быть использованы для проведения параллельных вычислений. В начале область подобных вычислений сводилась к реализации алгоритмов, необходимых для быстрого вывода изображений. Однако сейчас эта область существенно расширилась — современные графические процессоры позволяют проводить параллельные вычисления общего назначения, используя для этого операции с небольшими целыми и действительными числами. Это позволяет проводить исследования в различных областях математики, в частности, в области защиты информации.

В современных средствах защиты информации активное распространение получили вычисления в группе точек эллиптической кривой, определенной над конечным полем. Данные вычисления применяются в алгоритмах выработки и проверки цифровой подписи, в протоколах выработки общего ключа, при реализации схем гибридного шифрования и так далее, см. [2]. Основной проблемой, возникающей при использовании эллиптических кривых, является высокая трудоемкость реализации операции сложения и удвоения точек кривой.

В настоящей работе авторами рассматриваются вопросы реализации указанных операций с использованием графических процессоров. В начале мы реализуем базовые операции для конечного простого поля, а после, с их помощью, реализуем вычисления в группе точек эллиптической кривой. Параллельная архитектура графических процессоров позволяет рассмотреть большой класс алгоритмов и провести сравнительный анализ полученных реализаций. При вычислениях мы используем графическую платформу NVIDIA CUDA, см. [7], предоставляющую наиболее разнообразные возможности по реализации и оптимизации проводимых вычислений.

1 Эллиптические кривые

Пусть $p > 3$ простое число и $\mathbb{F}_p$ конечное поле из p элементов. Мы рассматриваем эллиптические кривые, заданные в аффинной форме сравнением

$$X_2^2 \equiv X_1^3 + aX_1 + b \pmod{p}, \quad 4a^3 + 27b^2 \not\equiv 0 \pmod{p}. \tag{1}$$

Пары (X_1, X_2), удовлетворяющие сравнению (1), называются точками кривой. Вместе с бесконечно удаленной точкой $\mathcal{O}$, они образуют абелеву группу, относительно операции сложения.

Операция сложения задается следующими соотношениями. Для любых двух точек $P_1(X_1, X_2)$ и $P_2(U_1, U_2)$, координаты которых удовлетворяют сравнению (1), выполнено

$$P_i + \mathcal{O} = \mathcal{O} + P_i = P_i, \quad i = 1, 2.$$

Если выполнены сравнения $X_1 \equiv U_1 \pmod{p}$ и $X_2 \equiv -U_2 \pmod{p}$, то полагаем $P_1 + P_2 = P_2 + P_1 = \mathcal{O}$. Во всех остальных случаях $P_1 + P_2 = P_3(Z_1, Z_2)$, где

$$Z_1 \equiv \lambda^2 - X_1 - U_1 \pmod{p}, \quad Z_2 \equiv (X_1 - Z_1)\lambda - X_2 \pmod{p} \tag{2}$$

и

$$\lambda \equiv \begin{cases} \frac{U_2 - X_2}{U_1 - X_1} \pmod{p}, & \text{если} \quad U_1 \not\equiv X_1 \pmod{p}, \\ \frac{3X_1^2 + a}{2X_2} \pmod{p}, & \text{если} \quad U_1 \equiv X_1 \pmod{p}. \end{cases}$$

При практических вычислениях, использование приведенных формул оказывается не эффективным, поскольку требует вычисления обратного элемента в поле $\mathbb{F}_p$. Для ускорения вычислений применяется проективная форма записи кривой (1)

$$x_2^2 x_3 \equiv x_1^3 + a x_1 x_3^2 + b x_3^3 \pmod{p}. \tag{3}$$

При этом, каждая точка кривой (3) является точкой проективного пространства $\mathbb{P}^2(\mathbb{F}_p)$. Напомним, что точку проективного пространства образует множество троек $(rx_1 : rx_2 : rx_3)$ для всех $r \in \mathbb{F}_p$ и $r \neq 0$. Переход от проективной точки к аффинной задается соотношением

$$(X_1, X_2) = \left(\frac{x_1}{x_3} \pmod{p}, \frac{x_2}{x_3} \pmod{p} \right),$$

при $x_3 \neq 0$. Тогда, все множество троек $(rx_1 : rx_2 : rx_3)$ соответствует одной аффинной точке (X_1, X_2).

Сравнению (3) также удовлетворяют тройки вида $(0 : r : 0)$, где r произвольный элемент поля $\mathbb{F}_p$. Для всех $r \neq 0$ множество таких троек определяет бесконечно удаленную точку $\mathcal{O}$. При $r = 0$ тройка $(0 : 0 : 0)$ не принадлежит проективному пространству $\mathbb{P}^2(\mathbb{F}_p)$.

Обратное соответствие, то есть переход от аффинной формы к проективной, задается очевидным равенством

$$(x_1 : x_2 : x_3) = (rX_1 : rX_2 : r), \quad r = 1, \ \ldots, p - 1.$$

Для эллиптической кривой, заданной в проективной форме (3), соотношения, определяющие групповой закон, могут быть переписаны в следующем виде. Пусть заданы две точки (x_1, x_2, x_3) и (u_1, u_2, u_3), удовлетворяющие уравнению (3), тогда их сумма (z_1, z_2, z_3) определяется, см. [5], следующими равенствами.

$$\begin{aligned} z_1 &= (u_3x_1 - u_1x_3)\left((u_1x_1(x_1u_3 + u_1x_3) + (u_3x_2 - u_2x_3)^2)u_3x_3 - (x_1^3u_3^3 + u_1^3u_3^3)\right), \\ z_2 &= u_3^4x_2(x_2^2x_3 + x_1^3) + x_3^4u_2(u_2^2u_3 - u_1^3) + u_2u_3^2x_3(3x_2^2x_3 - 2x_1^3) + \\ &\quad + u_1^2u_3x_2x_3^2(2u_1x_3 - 3u_3x_1) + 3u_2u_3^2x_3^2(u_1x_1^2 - u_2x_2x_3), \\ z_3 &= u_3x_3(u_3x_1 - u_1x_3)^3. \end{aligned} \tag{4}$$

Несмотря на столь громоздкий вид, приведенные соотношения могут быть эффективно вычислены с использованием равенств

$$z_1 = VA, \quad z_2 = U(V^2x_1u_3 - A) - V^3x_2u_3, \quad z_3 = V^3x_3u_3, \tag{5}$$

где $U = u_2x_3 - x_2u_3$, $V = u_1x_3 - x_1u_3$, $A = U^2x_3u_3 - V^2T$ и $T = u_1x_3 + x_1u_3$. Другим способом вычисления (4) могут служить, см. [2], соотношения

$$\begin{aligned} &\xi_1 = x_1u_3^2,\ \xi_2 = x_2u_3^3,\ \xi_3 = u_1x_3^2,\ \xi_4 = u_2x_3^2, \\ &\xi_5 = \xi_1 - \xi_3,\ \xi_6 = \xi_2 - \xi_4,\ \xi_7 = \xi_1 + \xi_3,\ \xi_8 = \xi_2 + \xi_4, \\ &z_3 = \xi_5x_3u_3, \\ &z_1 = \xi_6^2 - \xi_7\xi_5^2,\ \xi_9 = \xi_6^2 - 3z_1, 2z_2 = \xi_6\xi_9 - \xi_8\xi_5^3. \end{aligned}$$

В случае, когда точки (x_1, x_2, x_3) и (u_1, u_2, u_3) совпадают, результатом действия (4) будет нулевой вектор, не принадлежащий группе точек эллиптической кривой. В этом случае для вычисления $(z_1 : z_2 : z_3) = 2(x_1 : x_2 : x_3)$, нам необходимо воспользоваться соотношениями

$$\begin{aligned} z_1 &= 2x_2x_3(9x_1^4 + 6ax_1^2x_3^2 + a^2x_3^4 - 8x_1x_2^2x_3), \\ z_2 &= 36x_1^3x_2^2x_3 - 27x_1^6 - 27ax_1^4x_3^2 - 8x_2^4x_3^2 + 12ax_1x_2^2x_3^3 - 9a^2x_1^2x_3^4 - a^3x_3^6, \\ z_3 &= 8x_2^3x_3^3, \end{aligned} \tag{6}$$

которые могут быть эффективно вычислены при помощи соотношений

$$z_1 = 2SH, \quad z_2 = W(4F - H) - 8E^2, \quad z_3 = 8S^3,$$

где $S = x_2x_3$, $W = 3x_1^2 + ax_3^4$, $E = x_2S$, $F = x_1E$ и $H = W^2 - 8F$. Другим способом вычисления (6), см. [2], могут служить соотношения

$$\begin{aligned} &\xi_1 = 3x_1^2 + ax_3^4, \\ &z_3 = 2x_2x_3, \\ &\xi_2 = 4x_1x_2^2, \\ &z_1 = \xi_1^2 - 2\xi_2, \\ &\xi_3 = 8x_2^4, \\ &z_2 = 8\xi_1(\xi_2 - z_1) - \xi_3. \end{aligned}$$

Добавим, что все перечисленные выше соотношения вычисляются по модулю простого числа p.

Вычисление (4) и (6) сводится к вычислению некоторого числа операций сложения и умножения вычетов по модулю простого числа p. Таким образом, эффективная реализация элементарных операций в поле $\mathbb{F}_p$ является основой при вычислениях на эллиптических кривых.

Далее в работе мы рассмотрим два подхода к реализации элементарных операций сложения и умножения, учитывающих архитектуру графических вычислителей и возможность проведения параллельных вычислений,

Первый подход основан на представлении вычетов поля $\mathbb{F}_p$ в системе остаточных классов, то есть в виде множества остатков от деления по модулю некоторого набора составных чисел. Второй подход основан на разложении вычетов в системе счисления по основанию $W = 2^w$, для некоторого натурального w, определяемого особенностями графического вычислителя. Отметим, что оба подхода используют для умножения вычетов поля $\mathbb{F}_p$ операцию, введенную Питером Монтгомери. Такая операция является наиболее удобной, при проведении вычислений в проективных пространствах.

2 Арифметические представления

В начале опишем способ реализации элементарных операций сложения и умножения вычетов поля $\mathbb{F}_p$, предложенный Монтгомери в работе [6]. Данный способ позволяет несколько снизить трудоемкость вычислений. Это достигается за счет снижения трудоемкости операции приведения чисел по модулю простого числа p.

2.1 Умножение по Монтгомери

Зафиксируем натуральные числа m, r и $d \geqslant 2$, удовлетворяющие следующим условиям

$$p < m < r < 2p \quad \text{и} \quad (m-1)^2 < rp \tag{7}$$

$$\mathbf{НОД}\,(p,\ m) = \mathbf{НОД}\,(p,\ r) = \mathbf{НОД}\,(r,\ dm) = 1., \tag{8}$$

$$\mathbf{НОД}\,(d,\ m) = 1. \tag{9}$$

Мы предполагаем, что число d не превосходит по величине максимального целого, записываемого в одном регистре вычислительного средства; числа p, m, r — «большие», то есть требуют в своем представлении нескольких регистров вычислительного средства.

Определим отображение поля $\mathbb{F}_p$ в себя $\varphi : \mathbb{F}_p \to \mathbb{F}_p$

$$\varphi(\mathbb{F}_p) = \Phi = \{ar \pmod p \quad \text{для всех} \quad a \in \mathbb{F}_p\}.$$

Заметим, что поскольку p и r взаимно просты, то отображение φ обратимо и взаимно однозначно, то есть является изоморфизмом. Мы будем обозначать элементы множества Φ символами с верхней чертой, то есть $\overline{a} \equiv ar \pmod{p}$, $\overline{a} \in \Phi$. Введем на множестве Φ операции «сложения» и «умножения» элементов

$$\begin{aligned} \overline{a} + \overline{b} &\equiv (a+b)r \pmod{p}, \\ \overline{a} \cdot \overline{b} &\equiv (a \cdot b)r \pmod{p}. \end{aligned} \tag{10}$$

Легко видеть, что операция «сложения» элементов Φ совпадает с модульным сложением в $\mathbb{F}_p$. Для операции «умножения» это не верно, поскольку $\overline{a} \cdot \overline{b} \not\equiv (a \cdot b)r^2 \pmod{p}$.

При практической реализации операций на графической карте мы будем представлять элементы из множества Φ вычетами кольца $\mathbb{Z}_m$. Поскольку $m < r < 2p$, то каждому $\overline{a} \in \Phi$ будет соответствовать только один или два вычета $a' \in \mathbb{Z}_m$ таких, что $\overline{a} \equiv a' \pmod{p}$.

Для реализации операции «умножения» мы будем применять следующий алгоритм, основывающийся на идеях, предложенных П. Монтгомери в статье [6]. Предварительно, нам потребуется вычислить и сохранить в памяти две константы

$$\alpha \equiv -p^{-1} \pmod{r}, \quad \text{и} \quad \gamma \equiv r^{-1} \pmod{dm},$$

которые не зависят от перемножаемых значений.

Алгоритм 1: Алгоритм умножения

Вход : Параметры алгоритма p, m, r, d, α, γ и вычеты $\overline{a}, \overline{b} \in \mathbb{Z}_m$.
Выход: Вычет $\overline{w} \in \mathbb{Z}_m$, определяемый равенством $\overline{w} = \overline{a} \cdot \overline{b}$, см. (10).
1 Определить $t_m = \overline{a}\overline{b} \pmod{dm}$ и $t_r = \overline{a}\overline{b} \pmod{r}$.
2 Определить $q \equiv \alpha t_r \pmod{r}$.
3 Определить значение $s \equiv (t_m + qp)\gamma \pmod{dm}$.
4 **Если** $s \geqslant m$, **то** положить $\overline{w} = s - p$, иначе положить $\overline{w} = s$.

Лемма 1. *Приведенный алгоритм корректен и возвращает ожидаемый результат.*

Доказательство. На вход алгоритма подаются два вычета $\overline{a}, \overline{b} \in \mathbb{Z}_m$ такие, что $\overline{a} \equiv ar \pmod{p}$ и $\overline{b} \equiv br \pmod{p}$ для некоторых вычетов $a, b \in \mathbb{F}_p$. Мы ожидаем, что алгоритм вернет нам вычет $\overline{w} \in \mathbb{Z}_m$, удовлетворяющий сравнению $\overline{w} \equiv \overline{a}\overline{b}r^{-1} \equiv abr \pmod{p}$.

Обозначим символом t целое число, определяемое равенством $t = \overline{a}\overline{b}$, тогда выполнено неравенство $t \leqslant (m-1)^2$. Из сравнения $\alpha p \equiv -1 \pmod{r}$ вытекает, что

$$t + qp \equiv t + \alpha t_r p \equiv t_r - t_r \equiv 0 \pmod{r}.$$

Мы получили, что величина $s = \frac{t+qp}{r}$ является целым числом. Учитывая введенное нами ранее условие (7),получаем неравенство

$$t + qp < (m-1)^2 + rp < 2rp$$

из которого следует оценка на величину s сверху $s = \frac{t+qp}{r} < 2p < 2m$.

Рассмотрим равенство $t + qp = sr$. В силу того, что $\mathbf{НОД}(r,\ dm) = 1$ и $s < 2m \leqslant dm$, получаем, что выполнено сравнение

$$s = \frac{t+qp}{r} \equiv (t_m + qp)\gamma \pmod{dm}.$$

С другой стороны, выполнено сравнение $s = \frac{t+qp}{r} \equiv tr^{-1} \equiv ar \cdot br \cdot r^{-1} \equiv abr \pmod{p}$, из которого следует, что s является истинным значением, но не превосходящим величины $2m$.

Если $0 \leqslant s < m$, то это истинное значение и $\overline{w} = s$. Если же выполнено условие $s \geqslant m$, то $\overline{w} = s-p$. Поскольку $m < s < 2p$, то величина $\overline{w}$ положительна и не превосходит m. Лемма доказана. □

Следует сделать одно замечание о реализации последнего шага алгоритма «умножения». Определим величину $s_m \equiv s \pmod{m}$, тогда, полагая $s \leqslant 2m$, условие $s \geqslant m$ равносильно условию

$$s - s_m - m \equiv 0 \pmod{dm} \quad \text{или} \quad s - s_m - m \equiv 0 \pmod{d}. \tag{11}$$

Мы используем операцию умножения по Монтгомери для вычислений на множестве Ф. Результатом вычислений всегда является вычет, умноженный на некоторый фиксированный заранее множитель r.

Этот факт удобнее всего использовать при вычислениях с проективными координатами эллиптической кривой. В силу того, что решения $(x_1 : x_2 : x_3)$ и $(rx_1 : rx_2 : rx_3)$ сравнения (3) задают на кривой одну и ту же точку, нам не обязательно, после умножения по Монтгомери, проводить домножение координат точки на величину $r^{-1} \pmod{p}$.

2.2 Арифметика в остаточных классах

Для представления вычетов кольца $\mathbb{Z}_m$ мы будем использовать представление в остаточных классах или модульное представление целых чисел, см. [1, п. 4.3]. Подобное представление позволит нам реализовывать параллельные вычисления при реализации элементарных операций в кольце $\mathbb{Z}_m$. Мы будем считать, что выполнено равенство

$$m = \prod_{i=1}^{k} m_i, \quad \mathbf{НОД}(m_i,\ m_j) = 1 \quad \text{при} \quad i \neq j. \tag{12}$$

Согласно «китайской теореме об остатках», каждый вычет $\overline{a} \in \mathbb{Z}_m$ может быть однозначно представлен в виде вектора значений

$$\overline{a} = (a_1, \ \ldots, \ a_k), \quad \text{где} \quad \overline{a} \equiv a_i \pmod{m_i}, \ i = 1, \ldots, k.$$

Пусть $\overline{b} = (b_1, \ \ldots, b_k)$ произвольный вычет кольца $\mathbb{Z}_m$, тогда выполнены равенства, задающие арифметические операции

$$\begin{gathered}\overline{a} \pm \overline{b} = (a_1 \pm b_1 \pmod{m_1}, \ \ldots, \ a_k \pm b_k \pmod{m_k}),\\ \overline{a}\overline{b} = (a_1 b_1 \pmod{m_1}, \ \ldots, \ a_k b_k \pmod{m_k}),\\ (\overline{a})^{-1} = (a_1^{-1} \pmod{m_1}, \ \ldots, \ a_k^{-1} \pmod{m_k}).\end{gathered}$$

Рассмотрим вопросы параллельной реализации элементарных операций с вычетами по модулю p, представленными в виде элементов множества Ф. Перед тем, как описать собственно алгоритмы «сложения» и «умножения», мы приведем вспомогательный алгоритм, который может использоваться, в частности, для приведения вычетов по модулю простого числа p.

2.2.1 Смена системы остаточных классов

Пусть вычет $s \in \mathbb{Z}_m$, $0 \leqslant s < m$, представлен в остаточных классах в виде $s = (s_1, \ \ldots, s_k)$, где $s \equiv s_i \pmod{m_i}$, $i = 1, \ldots, k$. Мы можем представить его в другой системе остаточных классов, например $r = \prod_{j=1}^{l} r_j$, если для чисел m, r, выполнены условия (8).

Перед тем как описать алгоритм определим константы m_{ij}^{-1}, удовлетворяющие сравнениям

$$m_{ij}^{-1} \cdot m_i \equiv 1 \pmod{m_j}, \tag{13}$$

где $i, j = 1, \ldots, k$ и константы c_{ij}

$$c_{ij} \equiv \prod_{t=1}^{i-1} m_t \pmod{r_j}, \quad i = 1, \ldots, k, \quad j = 1, \ldots, l. \tag{14}$$

Смена остаточных классов описывается алгоритмом 2.

2.2.2 Алгоритмы в остаточных классах

Алгоритмы «сложения» и «вычитания» в остаточных классах совпадают с таковыми для позиционной системы счисления, с той лишь разницей, что элементарные операции над вычетами могут быть выполнены параллельно. Для определения переполнения по модулю m используется также вычет по модулю d. При использовании представления Монтгомери алгоритм не меняется.

Перед выполнением алгоритма «сложения» и «вычитания» необходимо определить следующие величины.

Алгоритм 2: Смена системы остаточных классов

Вход : Вычет $\overline{s} \in \mathbb{Z}_m$, заданный вектором $\overline{s} = (s_1, \ldots, s_k)$.
Выход: Вектор остатков $(\delta_1, \ldots, \delta_l)$ такой, что $\delta_j \equiv \overline{s} \pmod{r_j}$, $j = 1, \ldots, l$.

```
Положить вектор x = (x_1, ..., x_k) равным вектору s.
Для всех i = 2, ..., k выполнять
    Для всех j = i, ..., k выполнять
        вычислить x_j = (x_j - x_{i-1}) m_{i-1,j}^{-1} (mod m_j).
    конец
конец
Для всех j = 1, ..., l выполнять
    определить δ_j = (sum_{i=1}^{k} x_i c_ij) (mod r_j).
конец
```

1. Набор взаимно простых модулей $m_1, \ldots, m_k$ и число d такие, что $m = \prod_{i=1}^{k} m_i$ и выполняются условия (7 — 9).

2. Значение $m_d \equiv m \pmod{d}$.

3. Простое число p, по модулю которого проводятся вычисления, заданное в виде вектора $(p_0, p_1, \ldots, p_k)$, где $p_0 \equiv p \pmod{d}$ и $p_i \equiv p \pmod{m_i}$ для всех $i = 1, \ldots, k$.

4. Векторы $u_m = (u_0, u_1, \ldots, u_k)$ и $v_m = (v_0, v_1, \ldots, v_k)$ такие, что $u_0 \equiv u \pmod{d}$, $u_i \equiv u \pmod{m_i}$ и $v_0 \equiv v \pmod{d}$, $v_i \equiv v \pmod{m_i}$, для некоторых вычетов $u, v \in \mathbb{Z}_m$.

2.2.3 Алгоритм «умножения» в остаточных классах

Алгоритм умножения менее очевиден. Он также использует вспомогательное преобразование, используемое для смены остаточных классов. Перед выполнением алгоритма необходимо определить следующие величины.

1. Набор взаимно простых модулей $m_1, \ldots, m_k$,

   набор взаимно простых модулей $r_1, \ldots, r_l$,

   а также целое число d, удовлетворяющие условиям (7 — 9), при $m = \prod_{i=1}^{k} m_i$ и $r = \prod_{j=1}^{l} r_j$.

2. Задано значение $m_d \equiv m \pmod{d}$.

Алгоритм 3: Алгоритм «сложения» (вычитания) в остаточных классах

Вход : см. выше

Выход: Вектор $s_m = (s_0, s_1, \ldots, s_k)$ такой, что $s_0 \equiv s \pmod{d}$, $s_i \equiv s \pmod{m_i}$ для вычета $s \in \mathbb{Z}_m$ такого, что $s \equiv u \pm v \pmod{p}$.

1 Вычислить вектор $s_m = (s_0, s_1, \ldots, s_k)$, координаты которого удовлетворяют $s_0 \equiv u_0 \pm v_0 \pmod{d}$ и $s_i \equiv u_i \pm v_i \pmod{m_i}$ для $i = 1, \ldots, k$.

2 Используя алгоритм 2, вычислить s_d — остаток от деления числа $s_m = (s_0, \ldots, s_k)$ на целое число d.

3 **Если** $s_{m,0} - s_d = m_d$, **то** изменить координаты вектора s_m, вычисляя

$$\begin{aligned} s_{m,0} &\equiv s_{m,0} - p_0 \pmod{d}, \\ s_{m,i} &\equiv s_{m,i} - p_i \pmod{m_i}, \quad \text{при} \quad i = 1, \ldots, k. \end{aligned}$$

3. Набор значений $m_{i,j}^{-1}$, удовлетворяющий $m_i m_{i,j}^{-1} \equiv 1 \pmod{m_j}$ для всех $i < j \leqslant k$.

4. Набор значений $r_{i,j}^{-1}$, удовлетворяющий $r_i r_{i,j}^{-1} \equiv 1 \pmod{r_j}$ для всех $i < j \leqslant k$.

5. Простое число p, по модулю которого проводятся вычисления, заданное в виде вектора $(p_0, p_1, \ldots, p_k)$, где $p_0 \equiv p \pmod{d}$ и $p_i \equiv p \pmod{m_i}$ для всех $i = 1, \ldots, k$.

6. Вектор $\alpha = (\alpha_1, \ldots, \alpha_k)$, где $\alpha_i \equiv -p^{-1} \pmod{r_i}$ для всех $i = 1, \ldots, k$.

7. Вектор $\gamma = (\gamma_0, \gamma_1, \ldots, \gamma_k)$, где $\gamma_0 \equiv r^{-1} \pmod{d}$ и $\gamma_i \equiv r^{-1} \pmod{m_i}$ для всех $i = 1, \ldots, k$.

8. Два вектора $u_m = (u_{m,0}, u_{m,1}, \ldots, u_{m,k})$ и $u_r = (u_{r,1}, \ldots, u_{r,k})$ такие, что $u_{m,0} \equiv u \pmod{d}$, $u_{m,i} \equiv u \pmod{m_i}$ и $u_{r,i} \equiv u \pmod{r_i}$, $i = 1, \ldots, k$, для некоторого вычета $u \in \mathbb{Z}_m$.

9. Два вектора $v_m = (v_{m,0}, v_{m,1}, \ldots, v_{m,k})$ и $v_r = (v_{r,1}, \ldots, v_{r,k})$ такие, что $v_{m,0} \equiv v \pmod{d}$, $v_{m,i} \equiv v \pmod{m_i}$ и $v_{r,i} \equiv v \pmod{r_i}$, $i = 1, \ldots, k$, для некоторого вычета $v \in \mathbb{Z}_m$.

Собственно процедура умножения описывается алгоритмом 4.

Суммируя изложенные алгоритмы, заметим следующее. Для проведения вычислений на эллиптической кривой, по заданному простому числу p, нам необходимо определить константы m, r и d, удовлетворяющие (7 — 9). Как правило, константа d выбирается равной 2, а величины m и r — путем подбора маленьких простых, входящих в разложение m и r. В начале минимизируется величина

Алгоритм 4: Алгоритм «умножения» в остаточных классах

Вход : см. выше.

Выход: Два вектора $s_m = (s_{m,0}, s_{m,1}, \ldots, s_{m,k})$ и $s_r = (s_{r,1}, \ldots, s_{r,k})$ такие, что $s_{m,0} \equiv s \pmod{d}$, $s_{m,i} \equiv s \pmod{m_i}$ и $s_{r,i} \equiv s \pmod{r_i}$, $i = 1, \ldots, k$, для вычета $s \in \mathbb{Z}_m$ такого, что $s \equiv uvr^{-1} \pmod{p}$.

1 Вычислить вектор $t_r = (t_{r,1}, \ldots, t_{r,k})$, координаты которого удовлетворяют сравнениям $t_{r,i} \equiv u_{r,i}v_{r,i} \pmod{r_i}$ для $i = 1, \ldots, k$.

2 Вычислить вектор $q_r = (q_{r,1}, \ldots, q_{r,k})$, координаты которого удовлетворяют сравнениям $q_{r,i} \equiv \alpha_i t_{r,i} \pmod{r_i}$ для $i = 1, \ldots, k$.

3 Используя набор значений $r_{i,j}^{-1}$ и алгоритм 2, вычислить вектор $q_m = (q_{m,0}, q_{m,1}, \ldots, q_{m,k})$ — представление числа q_r в остаточных классах по модулям $d, m_1, \ldots, m_k$.

4 Вычислить вектор $t_m = (t_{m,0}, t_{m,1}, \ldots, t_{m,k})$, координаты которого удовлетворяют $t_{m,0} \equiv u_{m,0}v_{m,0} \pmod{d}$ и $t_{m,i} \equiv u_{m,i}v_{m,i} \pmod{m_i}$ для $i = 1, \ldots, k$.

5 Вычислить вектор $s_m = (s_{m,0}, s_{m,1}, \ldots, s_{m,k})$, координаты которого удовлетворяют сравнениям

$$s_{m,0} \equiv (t_{m,0} + q_{m,0}p_0)\gamma_{m,0} \pmod{d},$$
$$s_{m,i} \equiv (t_{m,i} + q_{m,i}p_i)\gamma_{m,i} \pmod{m_i}, \quad \text{при} \quad i = 1, \ldots, k.$$

6 Используя набор значений $m_{i,j}^{-1}$ и алгоритм 2, вычислить s_d — остаток от деления числа $s_m = (s_{m,1}, \ldots, s_{m,k})$ на целое число d.

7 **Если** $s_{m,0} - s_d = m_d$, **то** изменить координаты вектора s_m, вычисляя

$$s_{m,0} \equiv s_{m,0} - p_0 \pmod{d},$$
$$s_{m,i} \equiv s_{m,i} - p_i \pmod{m_i}, \quad \text{при} \quad i = 1, \ldots, k.$$

8 Используя набор значений $m_{i,j}^{-1}$ и алгоритм 2, вычислить вектор $s_r = (s_{r,1}, \ldots, s_{r,k})$ — представление числа s_r в остаточных классах по модулям $r_1, \ldots, r_k$.

$m-p$, а далее подбирается величина r, $m < r < 2p$. Отметим, что данные параметры подбираются один раз, до начала вычислений на эллиптической кривой.

Алгоритмы 3 и 4 используются для вычисления соотношений (4) и (6). Окончательное приведение координат вычисленной точки, то есть домножение на множитель $r^{-1} \pmod{p}$, не производится.

3 Позиционная система счисления

При представлении больших чисел в позиционной системе счисления в качестве основания выбирают $W = 2^w$, где w — число бит в машинном слове. Реализация модульной арифметики в позиционной системе счисления опирается на описанное выше представление Монтгомери [6]. Обзор основных схем реализации умножения с использованием преобразования может быть найден в работе [4].

Реализация позиционной системы счисления на платформе NVIDIA CUDA использует обработку с одним потоком на слово. Формулы (4) требуют место под хранение 9 промежуточных значений. Хранение этих данных в регистрах одного потока, на настоящий момент, возможно не более чем, для 128-битных чисел. Это ограничение дает нам оценку на величину числа p, т.е. $p < 2^{128}$. В реализации с одним потоком на слово промежуточные результаты хранятся в разделяемой памяти, что снижает требования из рассчёта на один поток. Вычисление итоговых значений битов переноса для операций сложений и вычитания реализовано через операцию параллельного вычисления префиксной суммы с оператором Stop-Generate-Propogate. Данный способ реализации сумматора с предсказанием переноса рассмотрен в [8]. Если положить $stop = 0$, $propogate = -1$ и $generate = 1$, то оператор SGP может быть эффективно реализован формулой

$$SGP(a, b) = (a\&b)|(a > 0), \tag{15}$$

где $\&$, $|$ и $>$ - операторы CUDA или языка C. Алгоритм вычисления префиксной суммы на NVIDIA CUDA использует обмен значениями через разделяемую память. Описание параллельного сложения приводится в алгоритме 5.

Алгоритм выполняется потоками одной вычислительной группы и синхронизации не требует. Дополнительно необходимо вычесть модуль из полученного значения, если произошёл перенос ($c_{n+1} > 0$) или полученное значение больше либо равно модулю. Для проверки последнего утверждения также используется вычисление префиксной суммы с оператором SGP и начальными значениями, определяемыми результатами попарного сравнения слов результата и модуля. Вычитание осуществляется аналогично, для него достаточно проверки на перенос из старшего разряда, при котором к результату необходимо прибавить значение модуля.

Наиболее полный анализ выбора схемы реализации модульного умножения на платформе NVIDIA CUDA дан в [3]. В настоящей работе при реализации

Алгоритм 5: Алгоритм параллельного сложения в позиционной системе счисления

Вход : n - число потоков, i - номер потока, a_i и b_i - слова, m - область разделяемой памяти с n элементами.

Выход: c_i - слово, такое что $c = (c_1, \ldots, c_{n+1})$ - сумма $a = (a_1, \ldots, a_n)$ и $b = (b_1, \ldots, b_n)$.

1 Вычислить значения суммы и переноса $(S, C) = a_i + b_i$.
2 **Если** S - *максимальное значение слова,* **то** $C = -1$.
3 Записать в общую память $m_i = C$.
4 Выполнить над массивом общей памяти m алгоритм вычисления префиксной суммы для оператора SGP.
5 Считать из общей памяти $C = m_i$.
6 Положить $c_i = S + (C > 0)$ для $i = 1, \ldots, n$ и $c_{n+1} = m_n > 0$.

была применена схема CIOS. Два внутренних цикла в схеме CIOS являются операций умножения s слов на одно слово с последующим сложением результата с $(s + 2)$ словами с получением результата в $(s + 2)$ слов. Умножение на одно слово может быть выполнено с помощью умножения по словам и параллельного сложения. Эта операция может быть объединена с операцией сложения, как показано в алгоритме 6.

Алгоритм 6: Алгоритм параллельного умножения на слово со сложением

Вход : n - число потоков, i - номер потока, a_i, b и c_i - слова, m - область разделяемой памяти с n элементами.

Выход: d_i - слово, такое что $d = (d_1, \ldots, d_{n+1})$ есть $a \times b + c$, где $a = (a_1, \ldots, a_n)$, b - слово и $c = (c_1, \ldots, c_n)$.

1 Вычислить $t_i = a_i \times b + c_i$ (результат - двойное слово).
2 Записать в общую память $m_i = LoWord(t_i)$ (младшее слово).
3 Положить $d_1 = m_1$.
4 **Если** $i = n$, **то** положить $q = 0$. Иначе, считать из общей памяти $q = m_{i+1}$.
5 Выполнить алгоритм 5:
$d_{i+1} = AlgAdd(n, i, HiWord(t_i)$старшее слово$, q, m)$ (старшее слово результата всегда равно 0, отбросить).

Алгоритм выполняется потоками одной вычислительной группы. Для корректной работы алгоритма, между шагами 3 и 4 требуется синхронизация, обеспечивающая возможность чтения результатов предыдущей записи в общую для всех потоков память. Теперь мы можем предъявить алгоритм 7 модульного умножения по схеме CIOS, использующий приведенный нами вспомогательный алгоритм 6.

При реализации данного алгоритма возможна оптимизация, позволяющая

Алгоритм 7: Алгоритм параллельного умножения по модулю

Вход : n - число потоков, i - номер потока, a_i, b_i, p_i и $p' = (-p^{-1} \pmod{2^{ns}})_1$ - слова (s - число бит в слове), m - область разделяемой памяти с n элементами, t - область разделяемой памяти с $(n+2)$ элементами.

Выход: c_i - слово, такое что $c = (c_1, \ldots, c_{n+1})$ есть $a \times b \pmod{p}$, где $a = (a_1, \ldots, a_n)$, $b = (b_1, \ldots, b_n)$ и $p = (p_1, \ldots, p_n)$.

1 Установить $t_i = 0$.
2 **Если** $i = 1$, **то** установить $t_{n+1} = t_{n+2} = 0$.
3 **Для всех** $j = 1, \ldots, n$ **выполнять**
4 Выполнить алгоритм 6: $t_i = AlgMad1(n, i, a_i, b_j, t_i, m)$.
5 **Если** $i = 1$, **то** вычислить значения суммы и переноса $(S, C) = t_{i+1} + d_{i+1}$ и записать $t_{i+1} = S$ и $t_{i+2} = C$.
6 Вычислить $M = LoWord(t_1 \times p')$ (младшее слово результата).
7 Выполнить алгоритм 6: $q = AlgMad1(n, i, p_i, M, t_i, m)$.
8 **Если** $i \neq 1$, **то** записать $t_{i-1} = q$.
9 **иначе** вычислить значение суммы и переноса $(S, C) = t_{n+1} + d_{n+1}$ и записать $t_n = S$ и $t_{n+1} = t_{n+2} + C$.
10 **конец**
11 **Если** $t_{n+1} = 0$ *или* $(t_1, \ldots, t_n) < (p_1, \ldots, p_n)$, **то** перейти к шагу 6.
12 Выполнить алгоритм вычитания: $t_i = AlgSub(n, i, t_i, p_i, m)$.
13 Положить $c_i = t_i$.

часть операций перенести в алгоритм 6, использовать один и тот же блок памяти для t и m и избежать в нём условия на шаге 4. Первая и последняя итерация также могут быть вынесены из цикла и совмещены с соседними операциями.

4 Результаты

Приведем результаты тестирования рассмотренных выше алгоритмов, реализованных на графических вычислителях NVIDIA CUDA с 16-ти битной и 32-х битной архитектурой. Вычисления проводились с использованием двух видеокарт — NVIDIA GTX 280 и GTX 580. Первая из видеокарт имеет CUDA capability 1.3, вторая — 2.0.

Дополнительно, приведем результаты реализации вычислений с использованием универсального процессора Intel Core i7-920. При вычислениях на центральном процессоре использовалась библиотека GMP, выполнявшаяся в режиме одного ядра (непараллельная версия). Добавим, что `pos` в имени реализации означает позиционную систему счисления, `rns` — систему остаточных классов. Приведенные в таблице числа обозначают количество операций сложения точек в секунду.

Реализация	Скорость	
`cpu-gmp`	1.3×10^6	
	GTX 580	GTX 280
`cuda-pos-16`	4.75×10^6	2.76×10^6
`cuda-pos-32`	21.9×10^6	9.49×10^6
`cuda-rns-16`	9.43×10^6	2.53×10^6
`cuda-rns-32`	21.14×10^6	3.64×10^6

Из приведённых данных очевидно, что 16-битные реализации не имеют преимуществ над 32-битными, в том числе и на видеокартах с отсутствием полной поддержки 32-битных операций CUDA capability < 2.0.

Дополнительное тестирование было проведено на другой машине с видеокартой NVIDIA GTX 550, имеющей CUDA capability 2.1. Результаты на ней повторяют таковые для GTX 580 с точностью до множителя. Таким образом, архитектурные изменения мультипроцессоров NVIDIA CUDA не вносят изменений в картину производительности различных реализаций.

5 Заключение

В настоящей работе мы изучили практические аспекты реализации алгоритмов сложения и удвоения точек эллиптической кривой на параллельных графических вычислителях семейства NVIDIA CUDA.

Для вычетов по модулю простого числа $p \sim 2^{128}$, мы использовали позиционное представление и представление в остаточных классах. Используя параллельные алгоритмы, мы реализовали элементарные операции модульной арифметики, применяемые при вычислениях на эллиптических кривых, и получили экспериментальные оценки количества реализуемых операций сложения точек эллиптической кривой в секунду.

Мы показали, что использование графических вычислителей, позволяет существенно увеличить скорость вычислений на эллиптических кривых, по сравнению с универсальными вычислителями. Основная причина этого, как и ожидалось, заключается в возможности параллельной обработки информации.

Мы экспериментально показали, что методы представления вычетов в остаточных классах не позволяют получить существенный выигрыш по времени по сравнению с представлением в позиционной системе счисления. Вместе с тем, как видно из приведенных в работе алгоритмов, реализация вычислений в позиционной системе счисления существенно проще.

СПИСОК ЦИТИРОВАННОЙ ЛИТЕРАТУРЫ

[1] Кнут Д.Э. Искусство программирования для ЭВМ. Получисленные алгоритмы. Том 2. — 3-е изд. — М.Виллиамс — 2000. — 788 стр.

[2] Blake I., Seroussi G., Smart N. Elliptic Curves in Cryptography. — Cambridge University Press. — 1999. — 205 pp.

[3] Henry R., Goldberg I. Solving Discrete Logarithms in Smooth-Order Groups with CUDA // CACR Tech Report 2012-02, http://cacr.uwaterloo.ca/techreports/2012/cacr2012-02.pdf

[4] Çetin Kaya Koç, Tolga Acar, Burton S. Laliski Jr. Analyzing and Comparing Montgomery Multiplication Algorithms // IEEE Micro — Volume 16, Issue 3, June 1996. — p. 26-33

[5] Koyama K., Tsuruoka Y. Speeding up elliptic cryptosystems using a signed binary window method // CRYPTO-92. — 1992. — p. 345-357.

[6] Montgomery P. Modular Multiplication Without Trial Division // Mathematics Of Computation. — Vol. 44 — №. 170 — 1985. — pp. 519-521.

[7] NVIDIA Corporation. NVIDIA CUDA C Programming Guide. — Version 3.2 — 2010.

[8] Tvrdik P. CS838: Topics in parallel computing. — 1999. — Available in Internet at http://www.cs.wisc.edu/t̃vrdik/21/ps/Section21.ps.

НИУ ВШЭ (МИЭМ)
Получено 02.05.2012