Вычислительно сложные криптографические задачи.

Н. Л. Замарашкин

ИВМ РАН им. Г.И. Марчука

Сложные задачи

(1) Multilinear Quadratic Problem: для квадратичных полиномов $f_1, f_2, \cdots, f_m$ от n переменных $x_1, x_2, \cdots, x_n$ над конечным $\mathbb{F}_2$ найти вектор $a = (a_1, a_2, \ldots, a_n) \in \mathbb{F}_2$ такой, что

$$f_1(a) = f_2(a) = \cdots = f_m(a) = 0 \in \mathbb{F}_2.$$

(2) RSA factorization: для заданного $n = pq$, где p и q неизвестные большие простые числа, определить p и q

(3) Вычисление спектра булевых функций: для функции $f : [-1, 1]^{2^n} \to [-1, 1]$ вычислить

$$\hat{f} = \mathcal{H}_n f,$$

где $\mathcal{H}_n$ – матрица Адамара

$$\mathcal{H}_n = \mathcal{H}_{n-1} \otimes \mathcal{H}, \quad \mathcal{H} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

Multilinear Quadratic Problem (MQP)

Для решения MQP предложены разнообразные методы:

1. агебраические методы
   1.1 XL метод (eXtended Linearization)
   1.2 методы на базисах Гребнера (методы Фожера F4, F5)
2. вероятностные алгоритмы (Локшанов, Bjorklund, Dinur)
3. SAT решатели (DPLL, CDCL)
4. „умный“перебор (Жу-Витсе, Буайге-Чен-Ченг-Чжоу-Нидерхаген)

Алгоритмы существенно различны. Эффективные реализации требуют вычислительных комплексов существенно разной архитектуры.

Multilinear Quadratic Problem (MQP)

Когда MQP является простой?

Имеется два очевидных частных случая:

1. когда $n \ll m$, а решение задачи дается перебором по 2^n значениям
$$\text{Complexity} = \mathcal{O}\left(2^n\right).$$
2. система f_i является линейной (все полиномы f_i имеют первую степень), а решение дается методом Гаусса за
$$\text{Complexity} = \mathcal{O}\left(\max\left\{m, n\right\} \min\left\{m, n\right\}^2\right).$$

Алгебраические методы решения MPQ сводят произвольную задачу к одному из (или к комбинации) легко решаемых частных случаев.

XL (eXtented Linearization) алгоритм

$$\begin{cases} x_1 + x_2x_3 = 1; \\ x_1 + x_1x_2 + x_2x_3 = 0; \\ x_2x_3 + x_2 = 1; \\ x_1 + x_3 + x_1x_2 + x_2x_3 = 0; \\ x_2 + x_3 + x_1x_2 = 0. \end{cases}$$

Переименуем мономы $y_1 = x_1,\ y_2 = x_2,\ y_3 = x_3,\ y_4 = x_1x_2,\ y_5 = x_2x_3$ и запишем в виде линейной системы

$$\begin{cases} y_1 + y_5 = 1; \\ y_1 + y_4 + y_5 = 0; \\ y_2 + y_5 = 1; \\ y_1 + y_3 + y_4 + y_5 = 0; \\ y_2 + y_3 + y_4 = 0. \end{cases}$$

В результате $y_1 = y_2 = 1$, $y_3 = y_4 = y_5 = 0$. Проверкой убеждаемся, что $a = (1, 1, 0)$ является решением.

XL (eXtented Linearization) алгоритм

Линеаризация – простая и эффективная идея, у которой имеется существенный недостаток.

XL (eXtended Linearization)

$$\begin{cases} x_1 + x_2x_3 = 1; \\ x_2 + x_2x_3 + x_3x_4 = 0; \\ x_3 + x_1x_2 + x_1x_3 + x_1x_4 + x_2x_3 + x_2x_4 + x_3x_4 = 0. \end{cases}$$

Прозводя линеаризацию, получим

$$\begin{cases} y_1 + y_8 = 1; \\ y_2 + y_8 + y_{10} = 0; \\ y_3 + y_5 + y_6 + y_7 + y_8 + y_9 + y_{10} = 0. \end{cases}$$

Линеаризация приводит к линейной системе с большим ядром

$$\dim(\mathrm{Ker}) \geq n - m = 7$$

Если $a_1, a_2, \cdots, a_7$ базис ядра, то число различных решений

$$\{\text{Число различных решений}\} = 2^{\dim(\mathrm{Ker})} = 2^7.$$

XL (eXtended Linearization)

Чтобы улучшить ситуацию, домножим первое уравнение на x_2, а второе на x_3, получим расширенную (extended) систему с меньшим ядром

$$\begin{cases} y_1 + y_8 = 1; \\ y_2 + y_8 + y_{10} = 0; \\ y_3 + y_5 + y_6 + y_7 + y_8 + y_9 + y_{10} = 0; \\ y_2 + y_5 + y_8 = 0; \\ y_{10} = 0; \end{cases}$$

$$\{\text{Число различных решений}\} = 2^{\dim(\mathrm{Ker})} = 2^5.$$

XL (eXtended Linearization)

Алгоритм XL

ВХОД: *Система из m полиномиальных уравнений от n неизвестных степени $D = 2$.*

ВЫХОД: *Решение (решения) системы полиномиальных уравнений.*

1. *Выбираем $\hat{D} > D$.*
2. *Составляем список $\mathcal{L}$ всех мономов степени не выше $\hat{D} - D$.*
3. *Составляем расширенную систему, домножая уравнения системы на все мономы из списка $\mathcal{L}$.*
4. *Линеаризуем полиномиальную систему.*
5. *Решаем линейную систему.*
6. *Восстанавливаем решение полиномиальной системы.*

Возникающая в XL алгоритме матрица расширенной линейной системы называется матрицей Маколея. Вычислительное ядро XL алгоритма – решение линейной системы с матрицей Маколея.

Свойства матрицы Маколея

Для MQP над $\mathbb{F}_2$:

- в наихудшем случае размер матрицы Маколея может доходить до 2^n, где n - число неизвестных
- число ненулей в произвольной строке не превосходит $n(n+1)/2$.

Матрица Маколея – большая разреженная матрица. Для эффективной реализации XL алгоритма требуется эффективный алгоритм решения систем линейных уравнений над полем $\mathbb{F}_2$ с большими разреженными матрицами.

DPLL алгоритм для MQP (как превратить MPQ в SAT)

$$\begin{cases} x_1 + x_2 x_3 = 1; \\ x_1 + x_2 = 1. \end{cases}$$

Составим соответствующую этой полиномиальной системе задачу SAT (ВЫПОЛНИМОСТЬ) в форме КНФ

$$\begin{aligned}
\text{SAT} &= (x_1 \Leftrightarrow (x_2 \wedge x_3)) \wedge (x_1 \Leftrightarrow x_2) \\
&= (x_1 \Rightarrow (x_2 \wedge x_3)) \wedge ((x_2 \wedge x_3) \Rightarrow x_1) \wedge ((x_1 \Rightarrow x_2) \wedge (x_2 \Rightarrow x_1)) \\
&= (\neg x_1 \vee (x_2 \wedge x_3)) \wedge (\neg(x_2 \wedge x_3) \vee x_1) \wedge ((\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_1)) \\
&= ((\neg x_1 \vee x_2) \wedge (\neg x_1 \vee x_3)) \wedge (\neg x_2 \vee \neg x_3 \vee x_1) \wedge ((\neg x_1 \vee x_2)) \\
&\wedge (\neg x_2 \vee x_1) \\
&= (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee \neg x_3 \vee x_1) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_1)
\end{aligned}$$

DPLL алгоритм для SAT

$$\begin{cases} C_1 : \neg x_1 \vee x_2 \vee \neg x_4 \\ C_2 : x_1 \vee x_3 \vee x_4 \\ C_3 : x_1 \vee \neg x_3 \\ C_4 : x_1 \vee x_3 \\ C_5 : x_2 \vee x_4. \end{cases}$$

Задача SAT состоит в поиске значений логичесаих переменных x_i, которые удовлетворяют всем условиям C_j.

SAT – первая задача, про которую было доказано, что она NP-полна. К этой задаче сводят:

- *задачи составления расписаний,*
- *верификация программного обеспечения,*
- *задачи оптимального управления,*
- *задачи биоинформатики.*

DPLL алгоритм для SAT

$$\begin{cases} C_1 : \neg x_1 \vee x_2 \vee \neg x_4 \\ C_2 : x_1 \vee x_3 \vee x_4 \\ C_3 : x_1 \vee \neg x_3 \\ C_4 : x_1 \vee x_3 \\ C_5 : x_2 \vee x_4 \end{cases}$$

Положим $x_1 = FALSE$ и рассмотрим все следствия:

1. C_1 удовлетворяется, поэтому исключается из рассмотрения
2. Упрощенная булева система принимает вид:

$$\begin{cases} \hat{C}_1 : x_3 \vee x_4 \\ \hat{C}_2 : \neg x_3 \\ \hat{C}_3 : x_3 \\ \hat{C}_4 : x_2 \vee x_4 \end{cases}$$

DPLL алгоритм для SAT

Положим $x_2 = TRUE$ и рассмотрим дальнейшие упрощения:

1. новая булева система принимает вид:

$$\begin{cases} \tilde{C}_1 : x_3 \vee x_4 \\ \tilde{C}_2 : \neg x_3 \\ \tilde{C}_3 : x_3 \end{cases}$$

Заметим, что, непротиворечивый выбор x_3 невозможен.
Возвращаемся (backtracing) к выбору x_2. Положим $x_2 = FALSE$

$$\begin{cases} \tilde{C}_1 : x_3 \vee x_4 \\ \tilde{C}_2 : \neg x_3 \\ \tilde{C}_3 : x_3 \\ \tilde{C}_4 : x_4 \end{cases}$$

Опять непротиворечивый выбор x_3 невозможен. Возвращаемся к выбору x_1. И так далее.

DPLL алгоритм для SAT

Описанный выше алгоритм решения задачи выполнимости в КНФ форме был предложен Davis-Putnam-Logemann-Loveland и получил название DPLL. Дальнейшее развитие DPLL алгоритма привело к семейству CDCL (Conflict Driven Clause Learning) алоритмов.

CDCL алгоритмы сохраняют ядро DPLL, но имеют

1. сложную процедуру анализа получаемых противоречий
2. как правило, нехронологический порядок возврата (основывается на эвристиках)

Вопросам построения эффективных паралельных реализаций для SAT задач занимаются более 20 лет. Современное программное обеспечение справляется с задачами, в которых число переменных порядка 10^6, а число условий $2 \cdot 10^7$. Рекордные размеры MQP задач составляют 80 уравнений с 80 неизвестными.

RSA шифрование

Пусть $n = pq$ с большими простыми p и q, а $x < n$ целое число, битовое представление которого является сообщением. Выберем число $e \in \mathbb{Z}^*_{(p-1)(q-1)}$. Число $y \in \mathbb{Z}_n$, получаемое по формуле,

$$y = x^e \mod n, \tag{1}$$

рассматривают как зашифрованное сообщение. Восстановление x не вызывает трудностей, если известен обратный к e элемент $d \in \mathbb{Z}^*_{(p-1)(q-1)}$. В этом случае

$$x = y^d \mod n. \tag{2}$$

В системе шифрования RSA параметры n и e считаются общедоступными. Определить d не сложно, зная p и q, но именно информация о делителях n является закрытой, а нахождение p и q относится к сложным задачам.

RSA числа

С 1990 года между научными группами ведется соревнование по факторизации RSA-чисел.

RSA-180	май 2010	И. Поповян (МГУ), А. Тимофеев
RSA-190	ноябрь 2010	И. Поповян (МГУ), А. Тимофеев
RSA-640	ноябрь 2005	F. Bahr, M. Boehm, J. Franke et. all
RSA-200	май 2005	F. Bahr, M. Boehm, J. Franke et. all
RSA-210	сентябрь 2013	R. Propper
RSA-704	июль 2012	S. Bai, E. Thome, P. Zimmermann
RSA-220	май 2016	S. Bai, E. Thome, P. Zimmermann
RSA-230	август 2018	Samuel S. Gross
RSA-768	декабрь 2009	P. Montgomery, A. Lenstra, E. Thome
RSA-232	февраль 2020	Д. Желтков, Н. Замарашкин, С. Матвеев
RSA-240	ноябрь 2019	E. Thome, P. Zimmermann et. all
RSA-250	февраль 2020	E. Thome, P. Zimmermann et. all

Коллеги

Дмитрий Желтков

Сергей Матвеев

Евгений Тыртышников

Вычислительные системы

Факторизация числа RSA-232 производилась на двух вычислительных кластерах:

1. Суперкомпьютер "Жорес" Сколковского института наук и технологий.
   Использовался для этапов:
   - Выбор полинома.
   - Просеивание.
   - Решение линейной системы над полем $\mathbb{Z}_2$.
2. Суперкомпьютер "Ломоносов" суперкомпьютерного центра МГУ им. М. В. Ломоносова.
   Использовался для этапа просеивания.

Алгоритм Ферма: $u^2 - v^2 = n$.

1. пусть

$$\begin{aligned} k &= \lfloor\sqrt{n}\rfloor \\ t &= 2k+1 \\ r &= k^2 - n \end{aligned}$$

2. пока r не является квадратом

$$\begin{aligned} r &= r+t \\ t &= t+2 \end{aligned}$$

3.

$$\begin{aligned} u &= (t-1)/2 \\ v &= \sqrt{r} \end{aligned}$$

Не любая пара u и v позволяет определить p и q.

Квадратичное решето: $u^2 - v^2 = 0 \mod n$

1. таких пар u, v существенно больше, чем в алгоритме Ферма, и более половины этих пар позволяют определить p и q.
2. рассмотрим отображение $f : \mathbb{Z} \to \mathbb{Z}$

$$f(r) = \left(r^2 - n\right) \mod n$$

3. назовем непустое множество $\mathcal{B}$ простых чисел базой делителей, а положительное целое число r гладким относительно выбраннной базы делителей [1], если

$$f(r) = \prod_{p_s \in \mathcal{B}} p_s^{\alpha_s}, \ \ \alpha_s > 0.$$

4. пусть $k = \lfloor\sqrt{n}\rfloor$; выберем $H \in \mathbb{Z}$ и обозначим через $\mathcal{R}_H$ все гладкие числа на отрезке $[k - H, k + H]$.

[1] J.D. Dixon. Asymptotically fast factorization of integers. Mathematics of Computation, 36: 255-260, 1981

Квадратичное решето

Заметим, когда $f(r)$ делится на простое p

$$f(r) = 0 \mod p,$$

то для любого $t \in \mathbb{Z}$

$$f(r + tp) = \left(\left(r^2 + 2trp + t^2p^2\right) \mod n\right) = 0 \mod p,$$

что дает основание называть алгоритм решетом и существенно сократить число реальных делений.

Разбивая отрезок $[k - H, k + H]$ на отрезки меньшей длины, можно искать гладкие числа на каждом из отрезков независимо.

Другими словами, в части поиска гладких чисел алгоритм обладает идеальной параллельной структурой, которая лежит на поверхности.

Квадратичное решето: $u^2 - v^2 = 0 \mod n$

Если $\#\mathcal{R}_H > \#\mathcal{B}$, то найдется $\Omega \subset \mathcal{R}_H$ и такие положительные целые числа β_s, что

$$\begin{aligned} u^2 &= \prod_{r_i \in \Omega} f(r_i) = \prod_{p_s \in \mathcal{B}} p_s^{2\beta_s}, \text{ где } 2\beta_s = \sum_{r_i \in \Omega} (\alpha_s)_i, \\ v^2 &= \prod_{r_i \in \mathcal{B}} r_i^2. \end{aligned}$$

Не сложно проверить, что

$$u^2 - v^2 = 0 \mod n.$$

Квадратичное решето: линейный этап

Составим матрицу $A \in \mathbb{Z}_2^{M \times N}$ с числом строк $M = \#\mathcal{B}$ и числом столбцов $N = \#\mathcal{R}_H$. Элементы a_{p_s, r_i} матрицы A определяются

$$a_{p_s, r_i} = a_{si} = (\alpha_s)_i \mod 2,$$

любое решение $x \in \mathbb{Z}_2^N$ однородной линейной системы

$$Ax = 0,$$

позволяет найти u^2 по формуле

$$u^2 = \prod_{i : x_i \neq 0} f(r_i).$$

Квадратичное решето: алгоритмическая сложность

Сложность алгоритма зависит от величины H, которая задает размер отрезка $[k - H, k + H]$ для поиска гладких чисел:

1. H должна быть достаточно большой, чтобы нашлось достаточно гладких чисел
2. с увеличением H пропорционально растет число проверок на гладкость

Оценка [2] сложности алгоритма

$$\mathcal{O}\left(\exp\left((1 + o(1))(\log n \log\log n)^{1/2}\right)\right).$$

---

[2] C. Pomerance. Smooth numbers and the Quadratic Sieve. Algorithmic Number Theory, MSRI Publication, 2008

Общее решето числового поля

Сложность алгоримта, по-общему согласию, оценивается как

$$\exp\left(\left(\left(\frac{64}{9}\right)^{1/3} + \mathcal{O}(1)\right)(\log n)^{1/3}(\log\log n)^{2/3}\right)$$

Применение алгоритма решета к RSA-232

Размеры баз делителей:

1. база алгебраических делителей $\mathcal{B}_a$ до 2^{36}
2. база рациональных делителей $\mathcal{B}_r$ до 2^{35}

Общее число гладких пар, полученное в результате алгоритма просеивания оказалось равным 5.7 миллиардов. Время, затраченное на поиск, ≈ 550 ядро-лет.

В результате работы алгоритма фильтрации (0.1 ядро-лет):

1. удаление дубликатов;
2. удаление синглтонов;
3. удаление клик;
4. выполнение слияний,

была получена матрица размеров 317 миллионов со средним числом ненулей в строке 170.

Линейный этап: матрица линейной системы

Матрицы линейных систем, возникающие после применения алгоритмов просеивания и фильтрации, обладают весьма необычными для вычислительной математики свойствами:

1. они крайне разреженные (170 элементов в строке длинной 370 миллионов)
2. расположение ненулей в матрице имеет случайный характер (это не дает возможности использовать стандартные процедуры предобработки матриц)
3. это системы над конечным полем, где отсутствует понятие о приближенном решении.

Сложность метода исключений Гаусса $\approx 10^{24}$ операций. Следовательно, даже для систем с 10^6 вычислительных ядер решение займет более 40 лет.

Линейный этап: портрет матрицы RSA-100

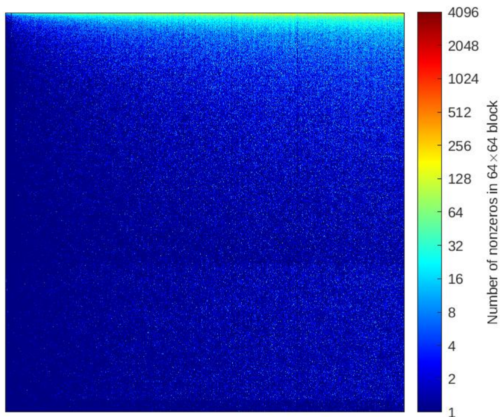


Рис. 2: Портрет матрицы RSA-100, полученный из общего решета числового поля и фильтрации

Портрет матрицы дифференциального оператора

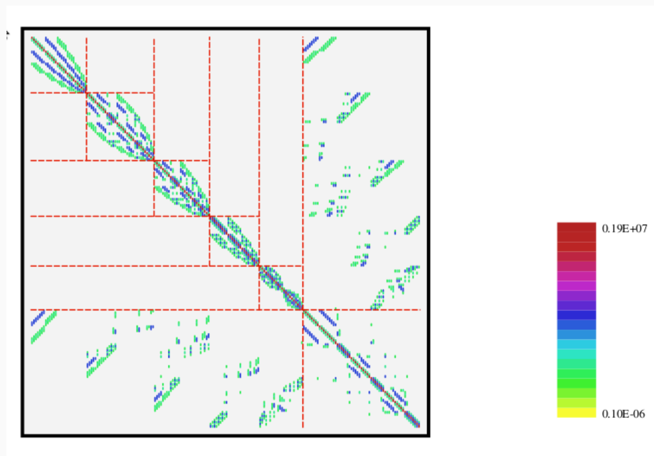


Рис. 3: Портрет конечноразностной аппроксимации дифференциального оператора

Разреженные системы над полем $\mathbb{R}$

Пусть

$$Ax = b, \quad A = A^T > 0 \in \mathbb{R}^{N \times N}.$$

Тогда стандартный метод Ланцоша записывается в виде

$$\begin{aligned} w_0 &= b; \\ w_i &= Aw_{i-1} - \sum_{j=0}^{i-1} c_{ij} w_j, \quad c_{ij} = \frac{w_j^T A^2 w_{i-1}}{w_{i-1}^T A w_{i-1}}, \end{aligned}$$

а решение x

$$x = \sum_{j=0}^{N-1} \frac{w_j^T b}{w_j^T A w_j}$$

Легко проверить, что в силу симметрии для последовательности w_i справедливы короткие соотношения

$$w_i = Aw_{i-1} - c_{i,i-1} w_{i-1} - c_{i,i-2} w_{i-2}.$$

Разреженные системы над полем $\mathbb{R}$

Важной особенностью крыловских методов над $\mathbb{R}$ является малое число итераций, необходимое для для получения хорошего приближения.

В случае $\mathbb{F}_2$

1. размерность крыловского пространства совпадает с размерностью задачи (отсутствует понятие приближенного решения)
2. с вероятностью $\frac{1}{2}$ выражение $w_i^T A w_i$ равно 0.

Чтобы получить работающий алгоритм Монтгомери

1. *ввел блочность* 64 *бит;*
2. *увеличил на* 1 *число слагаемых, в коротких соотношениях.*

Линейный этап: метод Ланцоша-Монтгомери [3]

$$\tilde{A}X = 0, \quad \tilde{A} \in \mathbb{Z}_2^{M \times N}, \quad M < N$$

Симметризуем систему

$$A = \tilde{A}^T \tilde{A} \in \mathbb{Z}_2^{N \times N}.$$

Выберем случайный $X_0 \in \mathbb{Z}_2^{N \times 64}$

$$AX_0 = V_0 = W_0 \in \mathbb{Z}_2^{N \times 64}, \text{ и } W_0^T A W_0 \text{ – невырожденная}$$

Далее

$$\begin{aligned} W_i &= V_i S_i, \\ V_{i+1} &= A W_i S_i^T + V_i + W_i C_{i+1,i} + W_{i-1} C_{i+1,i-1} + W_{i-2} C_{i+1,i-2}. \end{aligned}$$

[3] Montgomery, P. L. (1995). A Block Lanczos Algorithm for Finding Dependencies over GF(2). Lecture Notes in Computer Science, 106–120.

Линейный этап: метод Ланцоша-Монтгомери [4]

Матрицы S_i и $C_{i+1,i}$, $C_{i+1,i-1}$ и $C_{i+1,i-2}$ выбираются из тех условий, что

1. ${W_i}^T A W_j = 0$, если $i \neq j$
2. ${W_i}^T A W_i$ являются невырожденными
3. W_i образуют базис в пространстве Крылова $\mathcal{K}(A, V_0)$

Решение системы дается формулой

$$X = \sum_j W_j \left({W_j}^T A W_j \right)^{-1} {W_i}^T W_0,$$

с высокой вероятностью

$$\tilde{A}(X - X_0) = 0$$

[4] Montgomery, P. L. (1995). A Block Lanczos Algorithm for Finding Dependencies over GF(2). Lecture Notes in Computer Science, 106–120.

Линейный этап: сложность метода Ланцоша-Монтгомери

Пусть ρ обозначает среднее число ненулей в строке $\tilde{A}$. Отбросим все остальные вычисления в методе Ланцоша-Монтгомери и оставим только построение пространства Крылова $\mathcal{K}(A, V_0)$. Тогда наивно оценивая

$$\left\{ \begin{array}{c} \text{Время на} \\ \text{вычисления} \end{array} \right\} \geq 2\frac{\rho N^2}{64}/T \approx 2.9 \text{ ядро-лет}$$

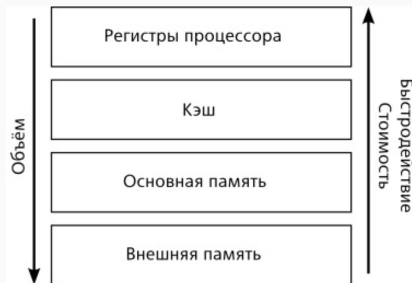
Однако примитивный алгоритм умножения матрицы на блок вида $N \times 64$ приведет к оценке времени в 450 ядра-лет, поскольку время работы будет определяться не временем T, выполнения одной арифметической операций, а пропускной способностью памяти.

Устройство памяти: иерархия

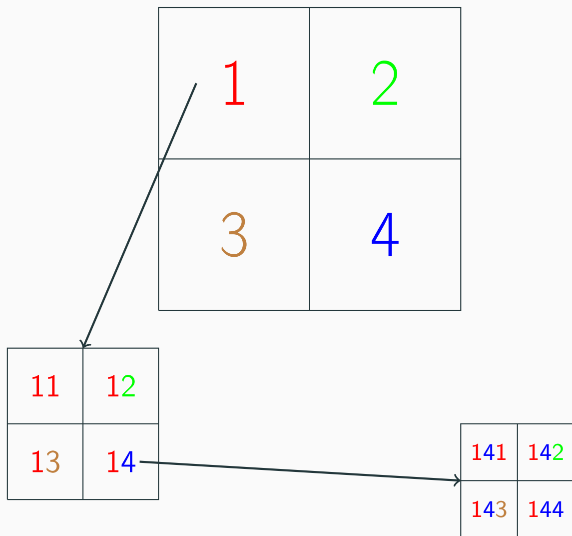
Память современных вычислительных устройств организована иерархически.

Память более высокого уровня в иерархии имеет:

- Выше пропускную способность.
- Ниже время доступа.
- Меньший объём.
- Выше стоимость (в пересчёте на бит).

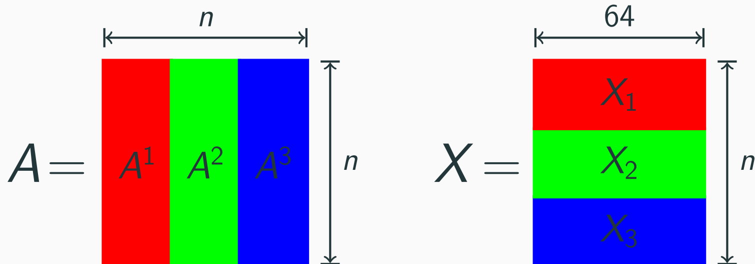


Кэш независимый алгоритм



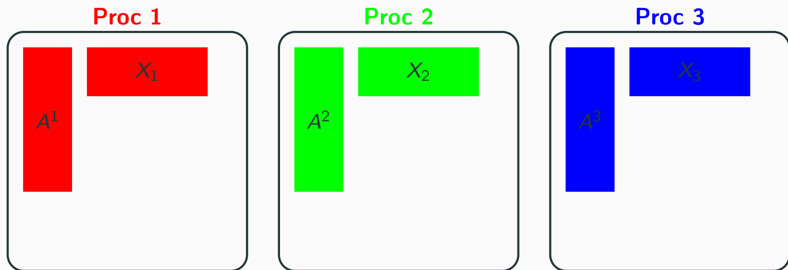
Время на построение пространства Крылова ≈ 36 ядро-лет.

Матрица на вектор: разбиение



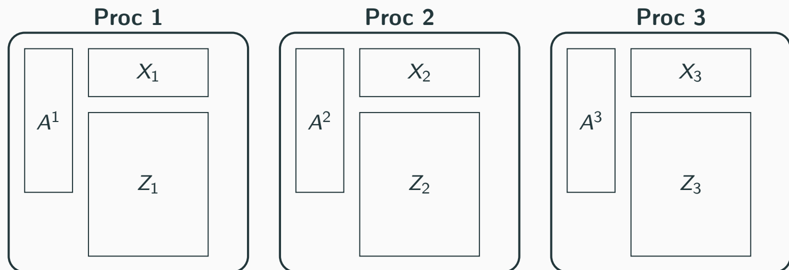
$K = 3$ – число вычислительных узлов

Матрица на вектор: распределение ресурсов по узлам



$K = 3$ – число вычислительных узлов

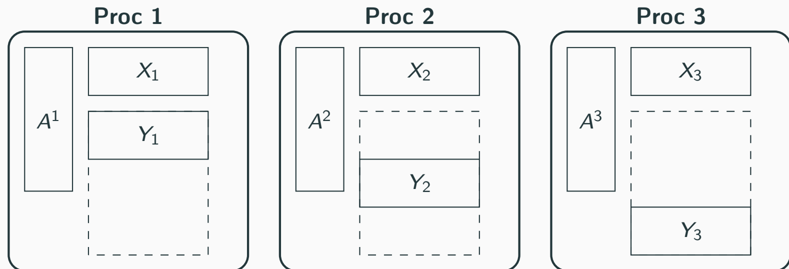
Матрица на вектор: умножение на узлах



Одновременное вычисление Z_i

$$Y = AX = A_1X_1 + A_2X_2 + A_3X_3 = Z_1 + Z_2 + Z_3$$

Матрица на вектор: циклические пересылки



$$Y = AX$$

Параллельная сложность

$$\left\{\begin{array}{c}\text{Время на}\\ \text{вычисления}\end{array}\right\} = N\left(\frac{|A|}{K} + N\right) = N\left(\frac{\rho N}{K} + N\right) = \mathcal{O}\left(\frac{\rho N^2}{K} + N^2\right)$$

$$\{\text{ Время на обмены}\} = N\left(\gamma N\left(1 - \frac{1}{K}\right)\right) = \mathcal{O}(\gamma N^2)$$

Таким образом, простая идея о перераспределении вычислений позволяет уменьшать время работы алгоритма, но приводит к появлению обменов и оставляет немасштабируемые слагаемые. Практика показывает, что эффективное число узлов для данного подхода $K < 50$, а, следовательно, только вычисление пространства Крылова заняло бы порядка 1 года.

Блочный метод Ланцоша-Монтгомери[5]

- короткие рекуррентные соотношения для $W_k \in \mathbb{F}^{N\times K\cdot 64}$

$$\begin{aligned} W_{k+1} &= AW_k + W_k C_k + W_{k-1} C_{k-1} + W_{k-2} C_{k-2}, \\ W_k^T A W_i &= 0, \quad \text{при } i \neq k \end{aligned}$$

- коэффициенты

$$C_k = W_k^T A^2 Q_k \left(W_k^T A W_k\right)^{-1}, \; C_{k-1} = W_{k-1}^T A^2 W_k \left(W_{k-1}^T A W_{k-1}\right)^{-1}$$

- решение

$$X_{\frac{N}{K}} = \sum_{i=1}^{\frac{n}{K}} Q_i \left(Q_i^T A Q_i\right)^{-1} Q_i^T B$$

Дает ли введение дополнительной блочности K преимущества в параллельной реализации?

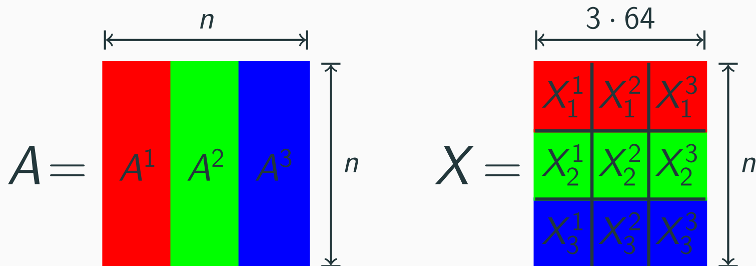
[5] Замарашкин Н.Л., «Алгоритмы для разреженных систем линейных уравнений над GF(2)»

Блочный метод Ланцоша-Монтгомери

Рассмотрим остальные операции в методе

1. сложность одного вычисления $X^T Y$ для $X, Y \in \mathbb{Z}_2^{N \times 64 \cdot K}$ оценивается как
$$\mathcal{O}\big(NK^2\big).$$
Всего таких вычислений $\mathcal{O}\left(\frac{N}{K}\right)$. Параллельная сложность не уменьшается с K.
2. сложность одного вычисления XF, где $X \in \mathbb{Z}_2^{64K \times 64K}$, а $F \in \mathbb{Z}_2^{N \times 64K}$ оценивается как
$$\mathcal{O}\big(NK^2\big).$$
Всего таких вычислений $\mathcal{O}\left(\frac{N}{K}\right)$. Параллельная сложность не уменьшается с K.
3. сложность F^{-1} растет как K^3 (параллельная сложность растет пропорционально K)

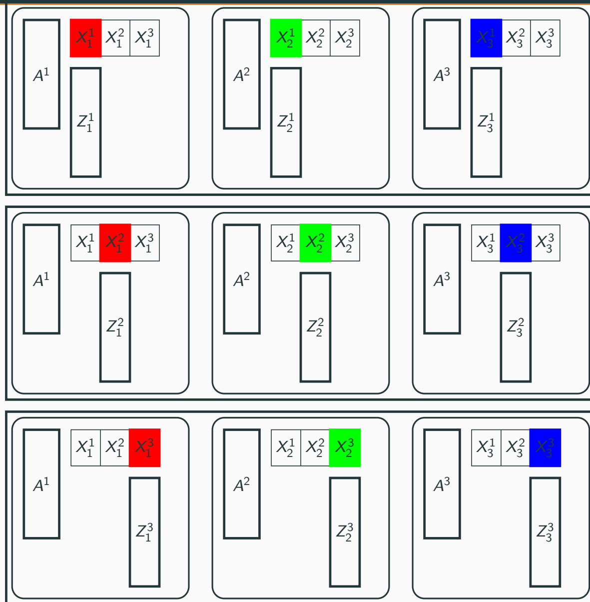
Матрица на блок: разбиение



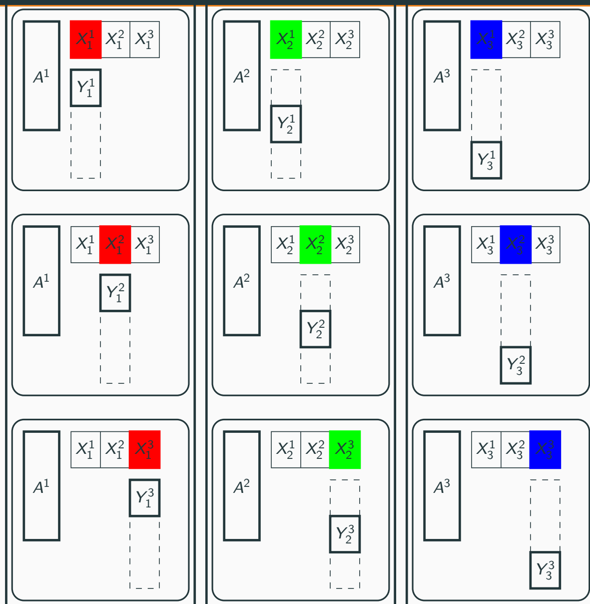
$$\left\{ \begin{array}{c} K_1 = 3, \text{ число блоков в матрице} \\ K_2 = 3, \text{ размер блока} \end{array} \right\} \rightarrow K = K_1 \cdot K_2$$

$K = 9$ – общее число вычислительных узлов

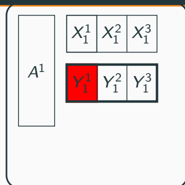
Матрица на блок: шаг 1

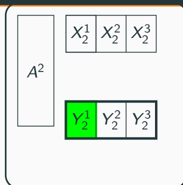


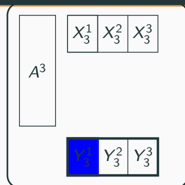
Матрица на блок: шаг 2

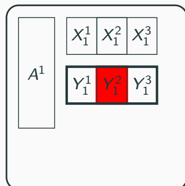


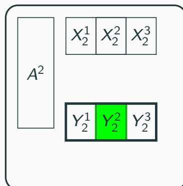
Матрица на блок: шаг 3

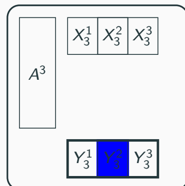


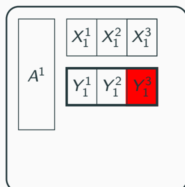


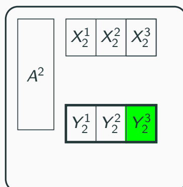


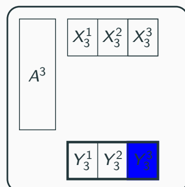












Параллельная сложность алгоритма

$$\left\{ \begin{array}{c} \text{Время на} \\ \text{вычисления} \end{array} \right\} = \frac{N}{K_2} \left(\frac{pN}{K_1} + N \right)$$

$$\{\text{Время на обмены}\} = \frac{\gamma N}{K_2} \left(N \left(1 - \frac{1}{K_1} \right) + \frac{N(K_2 - 1)}{K_1} \right)$$

1. K_1 – количество блоков столбцов в матрице
2. K_2 – количество векторов в блоке

Параллельная сложность алгоритма для случая $K_1 = K_2$

$$\left\{ \begin{array}{c} \text{Время} \\ \text{на вычисления} \end{array} \right\} = \mathcal{O}\left(\frac{\rho N^2}{K} + \frac{N^2}{\sqrt{K}}\right)$$

$$\{\text{ Время на обмены}\} = \mathcal{O}\left(\gamma \frac{N^2}{\sqrt{K}}\right)$$

Время на вычисления и обмены уменьшается (масштабируется) с увеличением числа узлов, но лишь как квадратный корень из общего числа вычислительных узлов.

Параллельная алгоритмическая сложность

Версия алгоритма, с помощью которой производилось разложение RSA-232, имеет существенно более сложную структуру:

1. матричное разбиение имеет место не только по столбцам, но и по строкам;
2. вычисления вида $X^T Y$, и XF ускоряются с помощью алгоритма В. Л. Арлазарова, Е. А. Диница, М. А. Кронрода и И. А. Фараджева (более известного, как алгоритм 4-х русских) и специального варианта метода Копперсмита.
3. используются специальные структуры данных и алгоритмы их преобразования с одного уровня быстрой памяти на другой для общей эффективной работы с памятью.

Для алгоритма имеется весьма точная оценка его параллельной сложности, из которой следует, что для большого числа вычислительных систем возможно добиться масштабирования вида $K^{\frac{2}{3}}$.

Алгоритм 4-х русских

$n = 2^p$.

$$A = \left[\begin{array}{c|c|c|c} & & & \\ & & & \\ A_1 & A_2 & \dots & A_{n/\log_2(n)} \\ & & & \\ & & & \end{array}\right],$$

$$B = \left[\begin{array}{ccccccc} & & & B_1 & & & \\ - & - & - & - & - & - & - \\ & & & B_2 & & & \\ - & - & - & - & - & - & - \\ & & & \dots & & & \\ - & - & - & - & - & - & - \\ & & & B_{n/\log_2(n)} & & & \end{array}\right].$$

Используя блочное представление для A и B, получаем

$$AB = \sum_{i=1}^{n/\log_2(n)} A_i B_i.$$

Алгоритм 4-х русских

В основе метода лежит возможность быстрого вычисления одного произведения $A_i B_i$.

Рассмотрим линейное пространство $\mathcal{L}_{B_i}$ строк матрицы B_i над полем $\mathbb{F}_2$. Из $\log_2(n)$ строк B_i можно составить n различных линейных комбинаций. Важное наблюдение состоит в том, что с использованием зеркального кода Грея возможно построить все n комбинаций ровно за n операций со строками.

Алгоритм 4-х русских

Пусть, например, $\log_2(n) = 2$ и $B_i = [b_1 b_2]^T$. Таким образом, в B_i две строки b_1^T и b_2^T. Линейные комбинации двух строк задаются парой бит. Запишем для каждой пары бит соответствующую линейную комбинацию

$$\begin{aligned} [00] &\rightarrow 0 \cdot b_1^T + 0 \cdot b_2^T = 0 \\ [01] &\rightarrow 0 \cdot b_1^T + 1 \cdot b_2^T = b_1^T \\ [10] &\rightarrow 1 \cdot b_1^T + 0 \cdot b_2^T = b_2^T \\ [11] &\rightarrow 1 \cdot b_1^T + 1 \cdot b_2^T = b_1^T + b_2^T. \end{aligned}$$

Строка с номером j в матрице A_i будет задавать номер в построенной таблице, и получение ответа не требует вычислений.

Битовая сложность умножения бинарных матриц порядка n этим алгоритмом $\mathcal{O}\left(n^3/\log n\right)$.

О проблеме реализации алгоритма 4-х русских

Таблица алгоритма одновременно является и его силой, и его слабостью. Чем больше размер таблицы, тем эффективне метод. С другой стороны, для того, чтобы ускорение было видно на практике, таблица должна лежать в быстрой части памяти.

Однако размер быстрой памяти сильно ограничен, а использование иерархии памяти по многим причинам затруднено.

Даже для матриц размера 2^{16} наилучшее получаемое нами ускорение (результат зависит от типа оборудования) не превосходит [6] значения 4, и, как правило, лежит в районе 2.5.

[6] Д.А. Желтков, З., 2018, 2019

Решето vs. линейный этап

RSA100	177638	8.7
RSA110	298165	7.5
RSA120	662264	8.2
RSA129	1230353	4.6
RSA130	1710081	2.5
RSA140	1865167	7.2
RSA155	3833856	7.1

RSA200	64000000	2.7
RSA220	132000000	13.7
RSA768	192796550	12.9
RSA240	282000000	9.5
RSA250	405000000	9.8

Сравнение линейных этапов RSA-232 и RSA-240

	RSA-232	RSA-240
Размер системы	317000000	282000000
Число ненулей в строке	170	200
Размер блока	8	4
Сложность	52 coreyears	69 coreyears

Таблица 1: Сравнение линейного этапа для RSA-232 и RSA-240

Замечание о задаче отображения алгоритмов на вычислитель сложной архитектуры

```
for i = 1, 2048
   for j = 1, 2048
      for k = 1, 2048
         c(i,j) = c(i,j) + a(i,k) b(k,j)
      end for
   end for
end for
```

Порядок циклов можно менять. Однако для разных комбинаций время исполнения меняется от 4 до 160 секунд. Более того, лишь незначительным преобразованием кода (рассматривая матрицы как блочные) возможно произвести умножение за 0.75 с.

Число арифметических операций одинаково для всех реализаций, разница в доступе к данным в памяти ЭВМ.