УДК 688.321

Р.В. Скуратовський

МОДИФІКОВАНИЙ АЛГОРИТМ ШЕНКСА З УПОРЯДКОВАНИМИ БЛОКАМИ

Majority of cryptanalytic methods can be modified due to parallel algorithms. One of them is the method of Shanks solving discrete logarithms problem. The main goal of this article is to construct algorithm, which allows parallel calculating all values from low and high pitch tables, to make this search more directed and to put in order all values of table elements. It will allow applying method of blocks searching, separation on the ordered sub-blocks, accelerate applying method of value indexing (or value hash). Parallel optimization and block parallel radix sorting, which became possible due to fast broadcast in duplex mode and mathematical models of algorithm, are the method of solving this problem. Method of parallel vector calculation, using values from BS table as coordinates, was proposed in this work. Optimal lengths of fine pitch and, as a consequence, high pitch were found for the method, which doesn't use complete order on the range of elements value of such low pitch. Method of improving Shanks algorithm was proposed.

Вступ

За часів виникнення метода Шенкса (1973 р.) для розв'язання проблеми дискретного логарифма його ефективність була незначною [1] з причини невисокої направленості перебору, який він використовує, і наявності великої кількості чисел для пошуку рівності. Це був один із перших методів, швидших, ніж метод прямого перебору. Його вдосконаленням займались такі відомі криптографи, як Л. Адлеман і А. Стеін [2, 3]. Завдяки розвитку комп'ютерної техніки з'явилася можливість удосконалення методу. В наш час, коли можлива паралельна обробка великих масивів інформації, його ефективність може зрости в стільки разів, скільки комп'ютерів ми застосовуємо, що можливо саме завдяки придатності методу до розпаралелювання обчислень, які в ньому проводяться. Однак поки що достатня увага цьому не приділялась.

Постановка задачі

Мета роботи – створити алгоритм із застосуванням паралельних обчислень, знайти його оптимальні параметри й оцінки складності обчислень.

Основні поняття

У роботі ми прагнемо зробити обчислення таблиць паралельним. Зменшенням часу на пошук рівності досягаємо впорядкування, що реалізується завдяки швидким пересилкам. Потім використовуємо пошук по лінійно впорядкованих множинах і по частково впорядкованих множинах. За основу для подальшої розробки взято класичний метод Шенкса [1]. Також необхідно порівняти затрати часу на пошук у невпорядкованому наборі масивів – табличках з методу Шенкса, і у впорядкованих блоках, які відповідають тим же табличкам, проте дещо іншого розміру.

Розв'язання проблеми дискретного логарифма (ПДЛ) за Д. Шенксом [1, 2] спирається на пошук рівностей у наборах чисел. У групі $\mathbb{C}_n$ обчислюються два ряди чисел:

$a, ga, g^2a, g^3a, \ldots, g^{m-1}a \pmod n$ – цей набір (таблицю) назвемо BS;

$g^m, g^{2m}, g^{3m}, g^{4m}, \ldots, g^{(m-1)m} \pmod n$ – цей набір (таблицю) назвемо GS.

Знаходять такі i та j, для яких $g^i a = g^{jm}$. Тоді $x = jm - i$, де $x = \log_g a$.

Остання рівність випливає з перетворень за модулем n:

$$g^x = g^{jm-i} = g^{jm}(g^i)^{-1} = g^{jm}aa^{-1}(g^i)^{-1} =$$

$$= g^{jm}a(g^ia)^{-1} = g^{jm}a(g^{jm})^{-1} = a$$

Перехід до $g^{jm}a(g^ia)^{-1}$ можливий, бо група комутативна. Отже, потрібно застосувати до методу Шенкса алгоритм із використанням паралельних обчислень і знайти параметри, за яких він має найбільшу обчислювальну ефективність.

Основні результати

Якщо додаткові параметри дають змогу, то до елементів g і a необхідно застосувати ізоморфізм $\varphi(g^k) = k \pmod n$ у групу $\mathbb{C}_n$, де груповою операцією є додавання за модулем n, яке є більш швидким, ніж множення. Всього

таких ізоморфізмів є $\varphi(n)$. Принцип паралельної обробки при цьому не зміниться.

Модель методу Шенкса для k-процесорного кластера без упорядкування лишків. Нехай маємо кластер з k нодами. Позначимо: m – величина малого кроку (BS). У звичайному методі Шенкса $m = \sqrt{n}$, де n – порядок групи, у якій розв'язується ПДЛ. У BS розподілимо елементи рівномірно між нодами, тобто на кожному комп'ютері буде обчислено $\frac{m}{k}$ елементів із таблиці BS (тобто з таблиці, яка відповідає малому кроку). Всього буде $\frac{n}{m}$ таблиць BS з m елементами в кожній, і вони будуть рівномірно розподілені на k комп'ютерах. Розмірність таблиці GS якраз дорівнює $\frac{n}{m}$.

Тоді кількість піднесень до степеня в BS дорівнює $\frac{n}{k}$. Далі маємо сумарну кількість кроків – елементарних обчислень у таблиці GS і в кожному екземплярі таблиці BS:

$$S: = \frac{n}{m} + \frac{m}{k}.$$

Наша задача – мінімізувати за величиною m суму S, тобто

$$\frac{n}{m} + \frac{m}{k} \to \min.$$

Для того щоб вибрати оптимальне m, подивимося на цю суму з точки зору нерівності Коші. Позначимо $a = \frac{n}{m}$, $b = \frac{m}{k}$. Як відомо, для довільних дійсних a, b виконується $a^2 + b^2 \geq 2ab$ чи $a + b \geq 2\sqrt{ab}$, а рівність досягається тільки при $a = b$. Отже, найменше значення, якого сума може досягти, – це $2\sqrt{ab}$. Тоді в нерівності $\frac{n}{m} + \frac{m}{k} \geq 2\sqrt{\frac{n}{k}}$ рівність має місце тоді і тільки тоді, коли $\frac{n}{m} = \frac{m}{k}$. Звідси $m^2 = nk$, $m = [\sqrt{nk}]$. Тоді загальна складність одночасного обчислення усіх таблиць становить $\frac{n}{\sqrt{nk}} + \frac{\sqrt{nk}}{k} = \frac{\sqrt{n}}{\sqrt{k}} + \frac{\sqrt{n}}{\sqrt{k}}$. Позначимо $l = \log n$, $m' = [\sqrt{nk}]$ – величина нового BS. Тоді в цілому на створення таблиць з урахуванням тривалості піднесення до степеня потрібно $O(m' l^2 / k) = O(\sqrt{nk}((\log^2 n)/k + \log k)) = O(\sqrt{n/k}(\log^2 n))$, $k << n$.

Висновок: малий крок $m' = [\sqrt{nk}]$ став більшим за рахунок розподілення обчислень, що є подібним до результату [3], де $m' = [\sqrt{nE_K}] = [\sqrt{n\alpha(g)L}]$ і, відповідно, $\frac{E_k}{m'}$ (в позначеннях [3]) великих кроків.

Модель із поблоковим упорядкуванням і пересилками. Спочатку запустимо передпроцес: якщо розв'язуємо ПДЛ у C_p, то необхідно знайти серед g^i $1 < i < \log_2 p$ такий, що вага Хемінга [4] $wt(g^i)$ є найменшою. Далі виражаємо g через g^i і застосовуємо метод для g^i, а потім знаходимо потрібний степінь для g. Лінійно впорядковані множини завжди є придатнішими для аналізу та порівняння, зокрема, до них зручно застосувати метод блочного пошуку. Доцільно застосувати метод паралельного сортування чисел на k комп'ютерах.

Крок 1. Нехай $m = \left[\frac{n}{k}\right]$. Розподілимо по m чисел на кожну ноду. На кожній ноді робимо сортування чисел, що містяться на ній.

Після обчислення кожною нодою табличок лишки за модулем p розподілені нерівномірно на нодах. Треба розбити всі лишки на множини

$$M_{i,i+1} = \{r: t_i < r < t_{i+1}, |M_{i,i+1}| = m\}$$

(множину $M_{i,i+1}$ назвемо i-м діапазоном – D_i). Якщо на j-й ноді лежить найбільша частина з множини $M_{i,i+1}$, то на цій же j-й ноді і збиратимемо всі елементи з множини $M_{i,i+1}$ (аналогічно на інших нодах). Тоді для збору $M_{i,i+1}$ на одній ноді, наприклад на j-й, кількість пересилок з кожного комп'ютера буде якомога меншою.

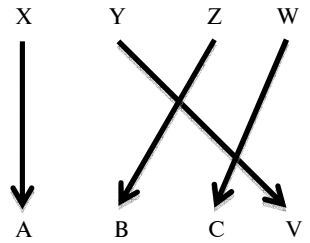
Крок 2. Знаходимо таке t_1, що кількість усіх чисел (на всіх нодах), які не більші t_1, становить рівно m. Це робиться за допомогою *двійкового пошуку*. Тобто вибір границь першого діапазону $[0; t_1)$ робимо, знаходячи t_1 методом половинного ділення до досягнення потрібної кількості m. А оскільки аргумент дискретний і лежить на відрізку довжини n, то пошук

займе $1+\log_2 N$ часу. Аналогічно знаходимо всі t_i, $1 < i \le k$.

Дводольна модель пересилок. У процесі пересилки кожна нода приймає і передає числа, але тільки від однієї ноди чи тільки одній ноді. Побудуємо модель пересилок у вигляді дводольного графа. Для цього продублюємо всі комп'ютери у вигляді другої частини графа з усіх комп'ютерів, але активними в цій частині є ті, які перебувають у стані прийому. А в першій (верхній) частині графа є ті ж самі ноди, однак активними з них є лише ті, які перебувають у стані відправки. Іншими словами, побудуємо модель у вигляді дводольного графа $K_{k,k}$, де k – кількість нод. У другій частині активні ті, які приймають, тобто в них входять стрілки; перша частина – ті, які передають, із них виходять стрілки.

Означення. Множину чисел, що лежить на *i*-й ноді і належить *j*-му діапазону, позначимо A_{ij}.

З'єднуємо *i*-ту ноду зверху з *j*-ю нодою знизу, на якій збираємо числа з D_j, якщо $A_{ij} \neq \varnothing$. Знайдемо паросполучення частин графа і зробимо сеанс пересилки, в якому на *j*-й ноді знизу висилаються числа з i_j нод зверху, де всі i_j різні – вони якраз і шукаються паросполученням. На *j*-ту ноду знизу приходять числа з *i*-го комп'ютера верхньої частини графа з діапазона D_j і з $M_{j-1,j}$, тобто вони з A_{ij}. На рисунку X, Y, Z, W – ноди з верхньої частини, а A, B, C, V – ноди з нижньої частини графа, де збираються лишки з відповідних діапазонів.



Паросполучення

Тоді для довільного i з $0 < i < k+1$ маємо $\sum_{j=1}^{k} \#A_{ij} = m_0$, якщо $n \vdots k$, або $\sum_{j=1}^{k} \#A_{ij} = m_0 + 1$, якщо $n \neq sk$, де # позначає потужність множини. Також для довільного j з $0 < j < k+1$ маємо $\sum_{i=1}^{k} \#A_{ij} = m_0$, якщо $n \vdots k$, або $\sum_{j=1}^{k} \#A_{ij} = m_0 + 1$, якщо $n \neq sk$.

Для кожного $i: 1 \le i \le k$ в лівій частині і довільного $j: 1 \le j \le k$ в правій при $m_0 > 0$ виконується

$$\sum_{j=1}^{k} \#A_{ij} = \sum_{i=1}^{k} \#A_{ij} = m_0 . \qquad (1)$$

За час сеансу пересилки на кожну ноду передається множина чисел

$$X = \min\{\#A_{i_1 1}, \#A_{i_2 2}, ..., \#A_{i_k k}\},\ i_j \neq j .$$

Тобто на кожний комп'ютер надійшло в сукупності X чисел з інших комп'ютерів.

Для k нод будуємо дводольний граф $K_{k,k}$. Для цього введемо на вершинах v з $K_{k,k}$ відношення суміжності: $\omega(v_i, v_j) = 1$ тоді і тільки тоді, коли $A_{ij} \neq \varnothing$, тобто з'єднаємо *i*-ту ноду зверху з *j*-ю нодою знизу ребром, якщо $A_{ij} \neq \varnothing$.

Лема. Паросполучення частин графа (або трансверсаль у матриці суміжності для $K_{k,k}$) існує, якщо виконується умова (1), яка характеризує відношення суміжності ребер дводольного графа.

Дійсно, довільні k нод мають переслати km_0 чисел, причому кожна передає і приймає рівно по m_0 чисел, тому прийняти за сеанс пересилки ці km_0 чисел можуть не менше ніж k нод. Ця умова виконується для кожного $1 \le j \le k$, тому умова $\#A' \le \#f(A')$ з наслідку 10.4.1 [6] виконана, де $f(A')$ – множина вершин, інцидентних ребрам, що виходять із множини вершин A'.

Позначимо відношення суміжності вершин у графі як $\omega(v_i, v_j)$, матрицю суміжності вершин дводольного графа як A.

Теорема. Якщо в матриці суміжності A замінити одинички в клітинах, де $\omega(v_i, v_j) = 1$, на значення A_{ij}, то отримаємо матрицю пересилок $\tilde{A}$ з нод нижньої частини графа в ноди верхньої частини, між якими існує трансверсаль без нульових елементів матриці.

Доведення. Оскільки для матриці $\tilde{A}$ виконується умова (1), то поділивши кожен її стовпець, отримаємо бістохастичну в традиційному розумінні матрицю, для якої застосовна лема і умови леми виконані. Отже, трансверсаль у матриці пересилок $\tilde{A}$, що відповідає пересилкам між комп'ютерами, існує, тому домноженням кожного коефіцієнта матриці A на суму пересилок по рядку m_0 ми встановлюємо відповідність між матрицями суміжності і матрицею пересилок. Обернену відповідність встановлюємо діленням на m_0. Отже, отримали взаємно однозначну відповідність між матрицями пересилок і матрицею суміжності. Це випливає з леми і доведено в [7].

Наслідок. Поки на кожній ноді існують числа, які потрібно переслати, то можлива одночасна пересилка з усіх нод навіть без використання дуплексного режиму обміну даними. Наявність же такого режиму тільки зменшує час на пересилку.

Назвемо матрицю бістохастичною, якщо вона квадратна, всі її елементи невід'ємні, а сума елементів у кожному рядку і в кожному стовпчику рівна одному і тому ж додатному числу.

Крок 3. Знаходження паросполучень з максимальною вагою.

3.1. Знайти перше паросполучення з максимально можливим мінімальним ребром.

Перше паросполучення доцільно вибрати таким $X = \max\limits_{T}\{\min\{\#A_{i_1 1}, \#A_{i_2 2}, ..., \#A_{i_k k}\}\}$, де T – множина всіх трансверсалей, щоб кожен комп'ютер зробив формальну пересилку самому собі якомога більшої кількості чисел. Час на цю пересилку і пошук такої комбінації ребер буде виключено із затраченого часу на пересилки, саме тому бажано, щоб їх час був найбільшим. Тобто це мінімум з максимально можливих пересилок, що будуть зроблені по петлі. Для того щоб знайти таке паросполучення з максимально можливим мінімальним ребром, слід використати динамічний алгоритм приєднання ребер [7], у порядку спадання їх ваги, у випадку неможливості реалізації часткового паросполучення з більшою кількістю ребер. Вийде $O(k)$ фаз збільшення потужності множини можливих у паросполученні ребер. Кожна фаза вимагає $O(k^2)$ проглядань різних ребер, а є k фаз [7], тому всього $O(k^3)$, що менше за $O(k^5)$ – час пошуку повним перебором із застосуванням алгоритму Куна. Але якщо при цьому використати алгоритм Хопкрофта–Карпа [7], то всього $O(k^2\sqrt{k}\log k)$. Хоча це не суттєво, бо $k << n$.

Далі можна шукати паросполучення в дводольному графі вже не з максимальним ребром серед мінімальних ваг ребер із різних паросполучень. Оскільки кількість пересланих чисел одна і та сама, то дбати про максимальність мінімального ребра вже не потрібно. Швидше знайти доповнений ланцюг з ребер, що буде новим паросполученням, модифікувавши старе паросполучення, що було зменшене на кілька ребер (зазвичай на одне ребро). Це можна зробити методом пошуку в ширину за $O(k^2)$, який описано нижче в адаптованому вигляді.

Після пересилки шукаємо нове паросполучення, знову з урахуванням зменшених множин A_{ij}, які позначимо A^1_{ij}: $|A^1_{ij}| = |A_{ij}| - X$. Це паросполучення знову буде знайдено згідно з лемою, бо матриця знову має властивість (1), оскільки сума по всіх рядках і стовпчиках цієї матриці однакова і не рівна 0. Дійсно, для довільного i з $0 < i < k+1$ маємо $\sum\limits_{j=1}^{k}\#A_{ij} = m_0 - X$, якщо $n : k$, або $\sum\limits_{j=1}^{k}\#A_{ij} = m_0 + 1 - X$, якщо $n \neq sk$. Також для довільного j з $0 < j < k+1$ маємо $\sum\limits_{i=1}^{k}\#A_{ij} = m_0 - X$, якщо $n : k$, або $\sum\limits_{j=1}^{k}\#A_{ij} = m_0 + 1 - X$, якщо $n \neq sk$. Зазначимо, що умова (1) після цього не порушена і матриця лишилася бістохастичною. Далі переходимо до нового сеансу пересилки.

3.2. Знайти нове паросполучення методом подовжуючого ланцюга. Подовжуючий ланцюг ефективно будується методом пошуку "в ширину" через модифікацію старого паросполучення.

3.3. Після завершення всіх пересилок числа будуть розподілені по відповідних діапазонах. Їх лишилось відсортувати методом порозрядного сортування чи індексувати на кожному комп'ютері, який містить рівно один діапазон D_i, використавши хешування.

Крок 4. Знайти рівність побітовим порівнянням, що буде найшвидшим у цьому випадку. Можна застосувати алгоритм [8] оптималь-

ного розбиття кожної таблиці індексів на блоки, за якого середнє число порівнянь, необхідних для пошуку елемента в індексі, є мінімальним. Якщо кількість елементів індексу рівна m, то оптимальний розмір блоку буде $\sqrt{m}$. При цьому число блоків і середня кількість порівнянь становить також $\sqrt{m}$.

Визначимо оцінку середньої кількості порівнянь біт чисел із різних таблиць для знаходження рівності. Введемо випадкову величину як кількість послідовно порівняних знаків α_i, β_i чисел до знаходження різних знаків, тобто $\alpha_j \neq \beta_j$, $1 \le i, j \le n$. Тобто спочатку порівнюємо перші знаки. Якщо вони збіглись, імовірність чого рівна $p_1 = \frac{1}{2}$, то порівнюємо другі знаки. Ймовірність одночасного збігу двох знаків у обох числах становить $p_1 = \frac{1}{4}$. Тоді

$$M(X) = 1 \cdot \frac{1}{2} + 2 \cdot \frac{1}{4} + 3 \cdot \frac{1}{8} + ... + (\log p_i^{-1}) p_i + ...$$

Відзначимо, що цей ряд є сумою геометричних прогресій:

$1 \cdot \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + ... = 1$ – така є одна;

$\frac{1}{4} + \frac{1}{8} + \frac{1}{16} + ... = \frac{1}{2}$ – таких є дві, але одну з них вже порахували;

$\frac{1}{8} + \frac{1}{16} + ... = \frac{1}{4}$ – таких є три, але дві з них вже порахували, і т.д.

У сумі матимемо 2. Отже, математичне сподівання числа порівнянь розрядів дорівнює 2.

Принцип пошуку доповненого ланцюга. Мета пошуку доповненого ланцюжка – знайти нову трансверсаль після занулення ребра з найменшою вагою. Позначимо перше побудоване паросполучення r_0, r_1 – часткове паросполучення, що утворилось з r_0 після вилучення ребра між v_0 і u_0 з мінімальною вагою в результаті пересилки, u_i – вершини нижньої частини графа, v_i – вершини верхньої частини. Нехай u_0 – вершина, що випала з паросполучення r_0. Добудуємо r_1 до повного паросполучення, починаючи з вершини u_0, за алгоритмом:

1) з можливих ребер вибираємо всі, які виходять з u_0. Далі рухаємося по кожному з них (пошук у ширину), потрапляємо в певну $u_{i_1}, ..., u_{i_j}$. Опишемо такий рух по одному з ребер (по інших аналогічно). Нехай потрапляємо в деяку v_{i_1};

2) переходимо з v_{i_1} в u_{i_1} по ребру з r_1, яке є ребром з r_0, і видаляємо це ребро з r_1;

3) беремо ребра, що виходять з v_{i_1}, і повторюємо процедуру, описану в п. 1;

4) якщо потрапили у вершину, для якої інцидентне ребро з r_1 утворене в п. 1, то цей ланцюжок перестаємо розглядати і розглядаємо інші ланцюжки, які таким же чином почали будувати з v_0;

5) як тільки потрапили в v_0, зупиняємо алгоритм і включаємо пройдені ребра з нижньої частини графа у верхню в нове паросполучення, а решту доповнюємо ребрами з r_0;

6) для пошуку подовженого ланцюга ефективно використати метод пошуку в ширину або метод пошуку в глибину. Тому складність пошуку подовженого ланцюжка – $O(e)$, де e – кількість наявних ребер. Ці методи доцільні, бо дають змогу добудувати редукований ланцюжок на одне ребро, щоб отримати нове паросполучення. Хоча можна заново побудувати ланцюг-паросполучення за методом Ахо–Хопкрофта.

Коректність. Нове r_1 обов'язково буде знайдене, воно існує за лемою 1, а тому його ребра мають входити і в u_0, і в кожну вершину, в яку ми потрапили на кроці 1 з u_0. Тому запропонований пошук, початий з вершини u_0, пройде, і по цій трансверсалі ланцюг не буде пропущеним. Довжина знайденого ланцюжка не може перевищити k, бо це довжина трансверсалі. Оскільки на кроці 1 і при кожному повторі кроку 1 ми могли потрапити, як максимум, у k вершин, то сукупна кількість кроків по ребрах рівна k^2. Отже, оцінка кількості пройдених ребер при пошуку – $O(k^2)$.

Для швидкого знаходження елементів таблиці BS можна застосовувати метод паралельного конвеєрного множення, обчисливши перед цим базові степені вигляду $g^{n/k}, g^{2n/k}, ..., g^{(k-1)n/k}$, де k – кількість процесорів. Для швидкого обчислення набору базових степенів доцільно використати паралельний метод бінарного піднесення до степеня й адитивні ланцюжки. В результаті складність обчислення всіх таблиць BS стано-

вить $O(\sqrt{n}\log^2 n/k)$; її можна покращити, застосувавши спеціальний перевибір елемента g. Для створення таблиць потрібно $O(ml^2/k) = O(\sqrt{n}(\log n \log g)/k), wt(g) < wt(n)$ елементарних операцій, більш точно це $O(ml \cdot wt(g)/k)$. Ця ж дія з використанням паралельного множення на основі багатополосних матричних операцій займе $O(ml \cdot \log(\log(wt(g)))/k)$, де $\log n$ – час на піднесення до степеня.

Після закінчення пересилок буде досягнуто відповідності один діапазон – одна нода. Залишається на кожній ноді лишки відсортувати методом половинного ділення чи індексувати на кожному комп'ютері, який містить рівно один діапазон і D_i. Для цього ефективно використати порозрядне сортування, бо числа мають однакову довжину і довжина чисел зростає разом з величиною масиву $O(\sqrt{n}\log n/k)$. Як альтернативний алгоритм можна застосувати алгоритм оптимального розбиття [8] кожної таблиці індексів на блоки, при якому середнє число порівнянь, необхідних для пошуку елемента в індексі, є мінімальним. Якщо кількість елементів індексу рівна m, то оптимальний розмір блоку буде $\sqrt{m}$. При цьому число блоків і середня кількість порівнянь також становлять $\sqrt{m}$.

Можливо також зробити пошук паросполучення r_1 заново, використавши для цього алгоритм Ахо–Хопкрофта $O(\sqrt{k}k^2)$ чи алгоритм Куна $O(k^3)$.

Ефективним є комбіноване застосування побудованого алгоритму з модифікованим Стінсоном алгоритмом Шенкса, що використовує розщеплювальну систему множин (*SS*), описану в [10, 4]. Його складність залежно від версії алгоритму – або $O\left(\binom{\lceil \log_2 n \rceil}{t/2}\right)$, або $O\left(N\binom{\lceil \log_2 n \rceil}{t/2}\right)$, де t – вага Хемінга дискретного логарифма елемента g, N – кількість блоків B_i в $(N; m.t)$-*SS*. Побудований алгоритм доцільно застосовувати, якщо $k << |B_i|$. В результаті застосування нашого алгоритму складність стане $O\left(\binom{\lceil \log_2 n \rceil}{t/2}/k\right)$ або $O\left(N\binom{\lceil \log_2 n \rceil}{t/2}/k\right)$, відповідно, необхідні обчислювальні ресурси $O\left(k\binom{\lceil \log_2 n \rceil}{t/2}\right)$. Також доцільно здійснювати паралельну обробку даних для різних вірогідних t. Зазначимо, що особливо ефективним метод *SS* є при t: $t/2 < [0,45\log_2 n]$, бо значення $C_n^{t/2}$ тим менше, чим більша різниця $n - t/2$.

Аналіз алгоритму і коректність роботи алгоритму. Після розбиття на діапазони і завершення їх наповнення однакові числа можуть існувати лише в межах одного діапазону, саме тому пошук пари збіжних чисел робиться не з множини $\#M = C_k^2\sqrt{n/k}\sqrt{n/k} = C_k^2 n/k$, а за допомогою сортування n/k елементів $O(\sqrt{n}\log n/k)$ і паралельної перевірки наявної рівності серед сусідніх елементів. Звідси економія кількості операцій на пошук збігів становить $C_k^2(n/k) - (n/k)$. А одне порозрядне порівняння [9] може в гіршому випадку зайняти $\log_2 n$.

Дійсно, при методі без упорядкованих діапазонів треба кожне число з таблиці GS величиною $m = [\sqrt{nk}]$, якому відповідає діапазон D_0, порівнювати з усіма іншими числами з усіх діапазонів, у т.ч. з самого D_0.

Складність методу впорядкованих блоків із пересилками. Позначимо: m – кількість чисел, l – довжина числа. Зрозуміло, що $l = \log n$, $m = [\sqrt{n}]$.

На створення таблиць треба $O(ml^2/k) = O(\sqrt{n}(\log n \log g)/k), wt(g) < wt(n)$ елементарних операцій, більш точно це $O(ml \cdot wt(g)/k)$. Порозрядне сортування: $O(ml/k) = O(\sqrt{n}\log n/k)$ елементарних операцій. Пересилка: $O(ml/k) = O(\sqrt{n}\log n/k)$. Тут слід врахувати, що максимальна кількість пересилок становить m^2, бо маємо m при кожному проходженні; в гіршому випадку відсилаємо $m-1$. Сумарна тривалість сеансу – m (бо з кожної ноди пересилається рівно m чисел).

Усього $O(\sqrt{n}\log^2 n/k)$. Порівняно з варіантом Стінсона [4, 10] цього алгоритму $O((N)C_{(l+1)/2}^{t/2})$, $l = [\log n]$, $0 < t < l$, $t, l \in \mathbb{C}$ є кращим для багатьох значень параметрів l і t. Таким чином, цей варіант методу Шенкса є дещо ефективнішим за складністю обчислень, ніж описаний у [12]. Хоча він теж вимагає великих витрат пам'яті, що є пропорційною до $\sqrt{n/2}$, але вже не потребує покрокового порівняння координат, що здійснюється по всіх точках, які зберігаються в памяті.

Отже, функція складності роботи алгоритму $f(n,k)$ обернено пропорційно залежить від k і за фіксованих значень параметрів $n, t = wt(g)$ має вигляд $f(n,k) = \frac{C}{k}$.

Складність методу невпорядкованих блоків без пересилок. Тут, як доведено, оптимальним є $m = [\sqrt{nk}]$. На створення таблиць треба $O(ml^2/k) = O(\sqrt{nk}(\log n \log g)/k + \log k)) = O(\sqrt{nk}(\log^2 n/k))$, $k << n, wt(g) < wt(n)$ множень. Порозрядне сортування: $O(ml/k) = O(\sqrt{nk} \cdot \log n/k)$.

Висновки

Метод паралельного обчислення впорядкованих блоків Шенкса має в $\sqrt{k}$ разів менше операцій, ніж його аналог із невпорядкованими блоками. Отже, збільшуючи число нод, можна значно підвищити ефективність. Порівняно з паралельним алгоритмом, який застосовує не менш ніж $(n^{2/3}t:(\log n)^{1/3})$ процесорів (де t – час на виконання множення), що мають одночасний доступ до спільної пам'яті [11], і який виконується не менш ніж за $O(n^{1/3}(\log n)^{4/3})$, запропонований алгоритм виконується не більш ніж за $O(\sqrt{nk} \cdot \log n/k)$, тобто при $k = (n^{2/3}t:(\log n)^{1/3})$ дає краще значення. Крім того, він не вимагає одночасного, постійного доступу процесорів до спільної пам'яті та не змінює закон оцінки складності і для $k < (n^{2/3}t:(\log n)^{1/3})$.

Перспективою подальших досліджень є комбінування цього методу з алгоритмом Поліга–Геллмана. Доцільно також розглянути модель пересилок на кілька комп'ютерів одночасно з одної ноди в дуплексному режимі з виконанням умови (1).

1. *Василенко О.Н.* Теоретико-числовые алгоритмы в криптографии. – М.: МЦНМО, 2003. – 328 с.
2. *L. Adleman,* "A subexponential algorithm for the discrete logarithm problem with applications to cryptography", in Proc. 20th Ann. IEEE Symp. Found. Comput. Sci., 1979, pp. 55–60.
3. *A. Stein and Teske E.,* "Optimized baby step-giant step methods", J. Ramanujan Math. Soc., vol. 20, no. 1, 2005, pp. 1–32.
4. *J.S. Coron et al.,* "A new baby – step gigant – step algorithm and some applications to cryptanalysis", in Proc. 7th Int. Conf. Cryptographic hardware and embedded system. Springer-Verlag, 2005, pp. 47–60.
5. *Основи* дискретної математики / Ю.В. Капітонова, С.Л. Кривий, А.А. Летичевський та ін. – К.: Наук. думка, 2002. – 578 с.
6. *Костюкова Н.И.* Графы и их применение. Комбинаторные алгоритмы для программистов. – М.: Изд-во ИНТУИТ.ру, 2007. – С. 231.
7. *Handbook* of Graph theory (Discrete Mathematics and its application), J.L. Gross and J. Yellen, Eds. Florida: CRS Press, 2004, 1192 p.
8. *Цегелик Г.Г.* Методы автоматической обработки информации. – Львов: Вища школа, 1981. – С. 132.
9. *Задирака В.К. Олексюк О.А.* Компьютерная арифметика многоразрядных чисел. – Тернополь: Підручники і посібники, 2003. – С. 247.
10. *D.R. Stinson,* "Some baby-step giant-step algorithm for low hamming weight discrete logarithm problem", Math. Comp., vol. 71, pp. 379–391, 2002.
11. *M.J. Wiener,* "The Full Cost of Cryptanalytic Attacks", J. Crypt., vol. 17, no. 2, pp. 105–124, 2004.
12. *Бессалов А.В., Телиженко А.Б.* Криптосистемы на эллиптических кривых. – К.: Політехніка, 2004. – 224 с.

Рекомендована Радою
Навчально-наукового комплексу
Інститут прикладного системного
аналізу НТУУ "КПІ"

Надійшла до редакції
3 червня 2013 року